

Houndtor: Histories & Legends Report Appendices

Gareth Wright
10239101



Table of Contents

Appendix A.....	4
Project Part one Deliverable PID:.....	4
Project Report 1 Deliverable Outline Report and initial research Document:.....	10
Part 2 deliverable report covering full research and.....	16
Appendix B.....	46
Dependencies.....	46
Class & Method.....	46
CameraSystem.....	46
Item.....	46
Object2Terrain.....	47
Image effects Complete.....	47
TriggerZone.....	47
TriggerObject.....	48
TriggerKittyAudio.....	48
SmoothCamFollow.....	48
Scroller2.....	48
GUIHelper.....	48
ExitBackUp.....	48
RotateObject.....	49
RB_FPS_Gruffy.....	49
PauseEvent.....	49
MouseLook.....	49
GenericTriggerScript.....	49
GUIMenuManager.....	50
GUIHoverControls.....	50
FindMyGPS.....	50
EndCreditsScript.....	50
GenericTrigger.....	51
Teleporter.....	51
LookAtTarget.....	51
System Overview.....	52
Appendix C.....	53

Class Definitions and Diagrams.....	53
Sequence Diagrams.....	63
Appendix D.....	66
XP Documented Game Code.....	66
Leveraged JavaScript's from the Purchased Particle game object.....	111
RayCast.js.....	113
PrefabGenerator.js.....	114
Appendix E.....	115
Appendix F.....	119
Appendix F.....	120

Appendix A

Project Part one Deliverable PID:

Introduction

History Mystery- An educational Exploration Adventure
(Dartmoor Edition)

This project has been put forward to deliver a working prototype of a full game concept based around the nuances of Dartmoor myth, legend and history.

The genre type will take influence from investigative and adventure based games, with predominant influences coming from adventure, mystery and investigative games that utilise 3D and interactive environments to move the story, such as the MYST™ franchise, The Mysterious Island™ franchise, the Darkfall™ franchise amongst others. The game itself will be focused on location centralised historical fact, folklore or Legend that surrounds areas of Dartmoor.

It will attempt to use the above stated focal nuances to develop a mystery exploration game that reveals educational facts, regarding localised Dartmoor history.

The final product would aim to deliver:

An unspecified number of unique experiences over the Dartmoor landscape, utilising high profile areas, such as the Dartmoor Tors, to drive the storyline.

Storyline utilising the game user for help in discovery of secrets. (The player takes on a challenge and attempts to solve mystery and puzzle elements that deliver learned or educational experiences to the user about historical, mythological, and legendary information about a location)

Photographic images of scenes designed around key Dartmoor locations used for puzzles/ investigative progression.

First Person Perspective view and control.

Appendices 1 Contents

These and other subject matters are later discussed in this document.

Table of Contents

Introduction	1
1 Project Proposal	3
2 Projects Components and Background	3
2.1 The Build	3
2.2 Subject Matter for Research	4
2.3 Desired Skills	4
2.4 Technologies available	4
2.5 Need for this type of development	5
3 Project Objectives	5
3.1 Initial Scope	5
3.2 Out of Scope	6
3.3 Contingency Plan	6
4 Project Log	6
4 .1 Blog	6
5 Investigation	7
6 Feasibility	7
6 .1 Purpose of the game	7
6 .2 Game scope	7
6 .3 Current deficiencies	7
6 .4 User Requirements	7
6 .5 T.E.L.O.S, as a feasibility tool	7
7 Analysis	8
8 Objectives	8
9 Approach to Methodologies	8
Works Cited	9

1 Project Proposal

Title: History Mystery- An educational Investigation Adventure- (Dartmoor Edition)

Format: Game/Educational interactive exploration/ experimental

Platform: Unity for prototyping, with further development to be undertaken in a suitable language.

Time Scale: 3 months for working prototype demonstrating conceptual premise.

Design Methodology: Agile **Conformance:** Lean (Lean Sensei) **Documentation:** UML

Lean is an adaptation of a set of principles derived from the Japanese Toyota Factory's set of principles and can be found detailed in the book entitled '*Toyota Production System, An Integrated Approach to Just-In-Time*' (1998). The term 'lean software development' was first coined in the book by (Poppendieck, 2003) who presented a set of principles

Development Methodology: Agile **Conformance:** eXtreme Programming **Documentation:** XP

Extreme programming can be seen to offer a developer a good avenue of system build approach with the emphasis on iteration and code redesign opportunities at the earliest possible identification and continued documentation of the computational system contained within the source code and relative "ReadMe" documents.

2 Projects Components and Background

2.1 The Build

The project will deliver, in the least, a working demonstrable prototype that will showcase the premise of the game and important aspects of the game mechanics.

The internal mechanisms the game based environment will be designed around and worked towards are:

First Person 3 Dimensional open-like environment.

Use of real photographic shots to deliver location based realism.

Photographs will be focused upon to deliver some puzzle elements as a gameplay factor.

Trail driven interaction game-play logic. (linear model for early prototypes)

Audio will play a relatively significant part in delivering the overall experience and, in order to compliment the project, a final year module centred on sound design has been undertaken in order that the correct aural accompaniments can be ascertained through better understanding of music and how to fit it into interactive scenarios for fullest effect.

The game will be developed on a Windows PC, but the prototype environment is Unity, which offers significant scope to deliver a prototype on other platforms simultaneously. (This is a future consideration and will be addendum to a final fully working product, which is currently considered out of scope for this project)

The control system will rely on mouse and key input. However, experimentation with control systems along the way may present alternative directions for player control mechanisms to be incorporated.

2.2 Subject Matter for Research

Games Research:

Myst franchise by Cyan™ Games

Return to Mysterious Island franchise by MC2 France (Microïds™)

The Adventure Company™, publishers of games such as Siberia, Broken Sword 2, Mysterious Journey, Aura: Fate of the Ages formally a division of developers DreamCatcher Interactive who sold the company on to Nordic Games in 2011 according to (Nordic Games Publishing AB, 2012).

Hardware Research:

Game controls and interaction options that may enhance or simplify the game-play experience.

System requirements needed from users computer in order to ascertain minimum requirements specification.

Design and Development Research:

Research into best practices, with respect to developing a game regarded as under the "adventure game" umbrella.

Research into applying educational techniques with games to present an experiential game-play that potentially offer the user useful chunk of storable information that they may take with them after finishing play.

Other Research:

Potential to transfer game model into a business model by which the game itself need only change by location and the parent model staying the same.

2.3 Desired Skills

During the final year of the degree, Unity has been undertaken as a prototype development environment. This coupled to some beginner experience using C# in an XNA environment has proven to be good choice as a prototyping tool. It is Unity 3D community Edition that will be used to

Seek to develop and hone programming skills with C#

Seek to understand better, the process of component oriented programming through Unity3D's environment.

Seek to develop and understand educational value in games and game playing which can be utilised as a key attribute in one's portfolio of ability.

Experience with using tangible real world elements mixed into a game environment (images, close representations of landscapes)

2.4 Technologies available

The following technologies will be used to progress the project, with respect to both documentation and physical development.

UML – to present the documentation of the system.

Trello Scrum boards to composite user stories and game ideas (mind dump).

Unity3D – For initial audio and game mechanics testing , later leading to developing a working prototype of the game that includes environmental and puzzle/ solution finding interactivity.

Blogger – Used to run a concurrent blog that attempts to less formally highlight the trials, tribulations and success` of the project in hand.

2.5 Need for this type of development

Though it is an opinion by the author of this document, it is firmly held that there is good reason to adopt the integration of current available technologies into tools that promote “edugaming” for two reasons:

To further cement the premise that game based applications have a place in the education sector and not only as a peripheral experience, but as a potential avenue to further expose “edugaming” as a plausible alternative to more orthodox or traditional methods.

This development may prove to have a place as a prototype that could be redesigned into a tourism based application where focus could shift to tourism related information through virtual experience prior to a visitation and the model that houses this premise could prove to have extended reach beyond this. However, this is matter for research

3 Project Objectives

3.1 Initial Scope

The project scope has itself a conformance to the lean principled methods of product development. Simply put, the scope may prove extensive and as such will be minimized to an interactive process by means of which can sufficiently demonstrate the potential the application might offer upon full completion.

The result will be a working prototype that can be utilised to demonstrate the concept, at the very least proof of concept.

The below points go forward to specify the expected scope of each element in terms of deliverables at any given stage:

Design- User stories & High level designs (ETC: 6 December 2012)

Development- Code & Testing (Concurrent with above timeline and extending into March 2013)

Full project delivery and documentation presentable DEADLINE: 31 March 2013

There has been a web accessible timeline created and is available @

<http://www.timetoast.com/timelines/initiation>

3.2 Out of Scope

The project will have limitations; the most pressing of which being time, and will undoubtedly present scheduling issues with respect to deliverables as a result. With this in mind, the maximum this project purports to deliver is a working prototype that can be interacted with as a demonstrative tool, all else will be considered out of scope and for future development unless reviewed (ad hoc) and is revealed as achievable in the remaining time frame.

With respect to the future, the development may have real world application as either an educational or tourism linked product which may present marketable features but, will also not be in scope for the final product deliverable this project requires.

3.3 Contingency Plan

If the investigation, analysis and early development were found to reveal, conclusively, that there is not enough substance in the concept of this prototype or adaptable use for this type of product, the game mechanics, environment, game models and theme can be altered to redeliver an experience that demonstrates an understanding of games concepts, but unfortunately may present nothing innovative.

4 Project Log

4.1 Blog

There will be a blog created over the coming weeks that offers a form of tracking the project's developments, though this should be viewed as informal, due to there being no strict guidance for how the blog should be presented or laid out.

The URL can be currently accessed through:

<http://mysteryedugame.blogspot.co.uk/>

5 Investigation

The investigation is currently underway with processes and findings recorded in the final document write up.

6 Feasibility

The feasibility study will use the TELOS acronym to further define the project idea as plausible and achievable. This type of documentation can be used to solidify expected outcomes and address issues like legality, technical and scheduled feasibility of the product.

Below follows an initial Feasibility surrounding the project

6.1 Purpose of the game

To provide a system that delivers an adventure, investigation led, experience by means of first person controlled environment themed on realistic localised areas of Dartmoor.

6.2 Game scope

This subject has been covered by section 3 entitled Project objectives but. This will be further investigated as to the potential of the product being extrapolated from its original concept to factory out a model that can be applied to other historical, mythical, and legendary areas of the UK.

6.3 Current deficiencies

The system will be built from scratch using Unity3D, C#, and a compliment of 3D modelling tools to produce content and other game assets. As such, the system has no identifiable deficiencies to highlight, due to it being a new development. This simply means there is a lack of a system reference model (an earlier version) to identify current deficiencies.

With respect to the development and prototyping environment, Unity3D

6.4 User Requirements

User requirements consist of:

Simple control mechanisms bearing in mind user cognitive capability, with respect to users being either young children or generalist computer users.

Encouraging/ coercive tutorial like instructions to ease user into using anything the game specifically requires one to be comfortable with. (For example: game keys or mouse actions)

Complimentary environment to add to the nuance of exploring the real thing.

Continual updates and inclusion of decisions based on user interactions concerning the game.

Approachable prototype deliveries incrementally presented during the later development/ test phases.

6.5 T.E.L.O.S, as a feasibility tool

A full TELOS breakdown will be delivered in the first project report of this assignment

7 Analysis

After investigations have concluded, an analysis will be undertaken to conclude the efficacy of the project and will go forward to provide conclusions and recommendations prior to entering the design stages of the system.

A critical analysis will be undertaken at the project's final steps to provide a comparison against what was aimed for and what was achieved.

8 Objectives

Conclusively, the Objective is to produce a complete document and working demonstrable prototype that meets the criteria laid out in this initiation document, and discovered requirement being laid out in future reports and project updates.

9 Approach to Methodologies

This section is also covered, in brief, at the beginning of this document under the section heading 1 "Project Proposal".

Below follows a definitive list of methodologies that the project aims to abide by, in order to deliver a complete document and prototype.

Agile Method:

Lean (Lean sensei)

This paradigm offers a good ability to produce relevant documentation and emphasis on iterative prototyping and sufficient room for change during development if required.

Documentation:

UML – User Stories, Class Diagrams and Relations, Sequence and Action diagrams to depict the system at the high level for non-technical audiences to gain an understanding of the product at the design level.

Scrum – Will be used to define sprint time constraints and deliverables during development and testing stages.

XP development documentation– Comprehensive code commenting that depicts the system from the technical perspective. Attempts to produce high levels of English explanation in the code may help to translate any technical aspects to those under the non-technical banner with an interest in the semantic functionality of the system.

Personal eXtreme Programming (PeP) – will be the main paradigm used to code the system, using aspects of TDD where system elements require it. This programming conformance is well suited to singular entity developments where both design and development can be conjoined by code comments for program documentation and light system designs using user stories and UML diagrams to describe the system at the high level.

Bibliography

Monden, Y. (1998). Toyota Production System, An Integrated Approach to Just-In-Time (Third edition ed.). Norcross: GA: Engineering & Management Press.

Poppendieck, M. (2003). Lean Software Development: An Agile Toolkit. Boston: Addison-Wesley Professional.

Project Report 1 Deliverable Outline Report and initial research Document:

PROC303 – FINAL YEAR PROJECT

Project Report Part 1

Draft documentation, Deliverables, and Early designs.

Gareth Wright 10239101

1.	Progress against plan	2
1.1	Overview	2
1.2	solidified idea, analysis and feasibility	2
1.2.1	Idea	2
1.2.2	Genre	3
1.2.3	Category	3
1.2.4	Platform	3
1.2.5	Play mechanics	3
1.2.6	Technologies	3
1.2.7	Target Audience	4
1.2.8	Key Feature/Unique Selling Point	5
1.2.9	T.E.L.O.S feasibility	5
2.	Demonstrable representations of the product	7
2.1	Initial Report Product deliverables/outputs	7
2.2	Development progress using Unity3D	7
2.3	Development Progress using stated 3D modelling tools	7
2.4	UI screens	8
2.5	Game Assets (Experimental and in-game placement models)	10
2.6	Prototype (Grey Box & Experimental Environment 1)	10
3.	Draft outline of the report.	11
3.1	Use Cases	11
3.2	High Level Action Diagram	13
3.3	Activity Diagram	14
3.4	Class Diagrams	15
3.5	Sequence Diagrams	15

1.	Progress against plan
1.1	Overview

After an implementation of the grey box for this development, it was found to be sufficiently informative to confirm it as a viable project and which will be deliverable by the 2013 deadline.

At this stage, the main presentation of any issues have come from steep learning curves in the 3D modelling software being used to complete the game assets. These programs are professional tools that the course provides no background on via modules related to the course and as such has had a costly effect on time in this project.

Though time constraints have been tight, the project as a whole has moved forward and, at the time of writing, is due to move into the second, asset complimented, grey box development testing phase. Once this phase has been instigated, development of in-game mechanics and logical flow will be implemented and applied to the test

environment. This is concurrent with the plan laid out by the project timeline, in fact, and at the time of writing, it is ahead of the schedule as grey box testing was immediately available in Unity with little needed to set up the scenario and researched and learned coding knowledge has proven useful to setup the in-game staples, for example; character controller, physics, simple terrain environments. The timeline is available at the following [http address](http://www.timetoast.com/timelines/initiation) for the reader to peruse- <http://www.timetoast.com/timelines/initiation>

Generally there was little research required to ensure legalities have and are kept within specified constraints, with respect to locational information to be extracted from Google Maps as a basis for the 3D terrain generation of the localised areas in-game. The legal issues arise mostly from printed or authored material that may prove useful to the narrative and storyline of the game premise. This further denoted an investigative requirement into the legalities surrounding the relay of information to children or minors through the game. As such, there is further research required for this element and is due to be undertaken in the coming weeks prior to the return to University in the New Year (2013).

1.2 solidified idea, analysis and feasibility

1.2.1 Idea

Mystery History is an explorative game designed to be both enjoyable to experience and educational by being historically and geographically correct alongside historic and myth or folklore as the driving force to entice user involvement.

The environment places the user in scenario whereby puzzles must be solved or clues found to complete the level and return to a selection screen that offers a range of destinations. These destinations are related by locality with this idea focusing around the locality of Dartmoor, using local history and folklore to inform the user about the area via playing the game.

1.2.2 Genre

This genre falls under the umbrella of Adventure games and shall be considered also as an educational game (or “Edugame”).

1.2.3 Category

Mystery History is designed to offer a unique experience of local areas through experiencing historic information about the locality through exploration, clue finding, and puzzle solving.

1.2.4 Platform

Nominated platform for delivery is Windows PC

1.2.5 Play mechanics

User will control movement in game through a first person perspective that will utilise keyboard key presses to engage motion, with mouse movement for looking around and mouse buttons to engage with the game puzzles, clue interactions, and navigating game texts.

UI elements will consist of the following:

- Gameplay Screen viewing game environment.
- Item obtainment indicator (thumbnail icon shows on screen if obtained).
- Text information boxes during game-play and for guidance/ help text.
- On-screen compass for directional reference in game.

1.2.6 Technologies

Unity 3D Engine Integrated Development environment:

This robust sandbox environment poses an ideal prototyping and experimental development scenario. It has richly populated community resources that can be called upon to speed up some initial processes that can be refined at a later stage for example; Character controllers, particle-effects components, gravity and physics management.

Sketch-up 8 Community edition:

This is a 3D modelling package that has now become established enough to offer simpler model creation in sketch up with easy exportation to other 3D modelling software like 3dsMax and Maya by Autodesk™, or the open-source software Blender.

Autodesk™: 3DS Max, Maya & Soft Image

The above 3D modelling software tools are part of a suite of professional standard tools used in industry. (Autodesk Inc, 2012)

Blender™:

An open-source, community contributed, 3D modelling software package popular amongst independent developers, first and foremost because it is free to use, but also due to its friendly nature and intuitive UI.

1.2.7 Target Audience

The target audience is aimed towards children and a wide range of interested parties looking to enjoy receiving factual information and local lore as part of an interactive exploration experience.

There is scope to extend the target to tourism based with this product, with respect to identification of other target audiences.

1.2.8 Key Feature/Unique Selling Point

The unique selling point of this, fundamentally, game is that the content is offered as historically correct and delivered through the mechanics of gameplay as an educational experience during, and hopefully afterwards too, user engagement.

Another Unique selling point is the localisation of the game environments, which are based on real locations around Dartmoor. This shall be achieved by obtaining maps via Google™ products and converting them through a process of different modelling software to a 3 dimensional terrain for use in the game. The representation should deliver a good level of accuracy once completed true to the existing real life terrain of the locality.

The ability for the player to be relatively free during game play to explore the local environment but, with a task at hand, per environment, to encourage exploration/ investigation of the game's offerings.

1.2.9 T.E.L.O.S feasibility

Technical

With respect to technical expertise required to deliver the product as a prototype built in Unity3D™, sufficient knowledge has been obtained to instigate initial grey box testing, this allows for fast experimentation and further familiarising one's self with the development environment for the game.

Through the further use of purchased literature, online documentation, and forum support, technical capability as a feasibility measure is in place and developing alongside the project's progression.

Economic- Cost / Benefit Analysis

- Cost

The cost of this project is solely time based but. This does not mean it is any less accountable to analysis of feasibility. As such, time constraints have proven typically tight against the scheduled stages of the project timeline and adjustments had to be made to reconcile an achievable outcome. These changes, identified in the conclusion to this feasibility study, have not affected the demonstrable ability of the final prototype, rather only affected the opportunity to develop the game outside of the Unity3D™ engine sandbox.

Another time constraint that should be considered a cost, with respect to the projects requirement of input, are concurrent projects that assist in the unbalancing of meeting the deadlines and cut-off points this project has and will continue to present.

Operational costs can be attributed to users of the product, but it is an assumption that the user will have a sufficient Windows platform PC in order to use the product.

- Benefit
 - o The benefit to this development is of educational value and will deliver an experience that may assist in divulging information relevant to national curriculum history.
 - o Further benefit to the Dartmoor tourism industry may be gained by this development by increasing locality awareness amongst users, increased chance of visits to local tourist spots through information received by the user in-game.

Legal

- o IP conflicts: None
- o Non-original game assets: As required by owner
- o Google Maps as a source of accurate data: There was found to be no conflicts at this time related to using the Google Maps application to create a reference map for building upon a geologic terrain. This is due to the map becoming changed from its original flat 3D state to a 3D interpreted state and, furthermore, is used under the fair use act for non-profitable purposes.
- o Data protection act stature is still under research, with respect to use of historical information by citation or anecdotally. This important aspect must be fully covered to ensure use of historic and folklore data in the game product can be used and remains accurately relayed.

Operational

This system is not a system that is built to solve a problem but, is more inclined to present a novel method of educational digestion through game-play. The problem in hand could be surmised, at best, to be that education does not reach all learners equally and that opportunity to deliver pertinent historic content themed for engagement and enjoyment may also help to improve the retention of information by familiarisation to the experience along the way.

Scheduling

The project schedule was found to have remained feasible and realistic against completion of the project overall. At the time of writing, the schedule has continued to demand time but has also offered moments of significant progress, with respect to developing a grey box in a matter of weeks for the game idea, whereas this had been scheduled for late December at the earliest, alongside class structures in place for adding code to the next iteration of prototype. These have currently been superseded by the quick fashion in which Unity offers usable results.

2. Demonstrable representations of the product

2.1 Initial Report Product deliverables/outputs

Deliverable	Output
Use Cases and initial Research Documentation	See section2 sub-section 2.5
Grey Box Experimental Environment in Unity3D	See Attached Compiled Grey Box demonstrable
Assets	See attached Grey box demonstrable for current prototyped asset experimentations

2.2 Development progress using Unity3D

Unity3D has many advantages, first and foremost for this project; it has proven invaluable as a testing and experimental environment. There is always much to learn from this sandbox environment and as such continual aggregation of knowledge is constantly being acquired at every session. This has been complimented by initial user

training gained through concurrent modules to the course overall, under the module titles AINT304 and AINT 306 it has been possible to explore further intricacies on offer from the Unity development environment and a chance to develop some coded mechanisms that will be utilised by the game product's final version.

2.3 Development Progress using stated 3D modelling tools

3D modelling has been found as a difficult process to undertake due to the nuance of understanding the complex UI these software packages often present. With this in mind, the software package Blender™ has been used to effect basic models without textures, normal or UV mapping applied.

The modelling software by Google – SketchUp™ will be used to create the terrain environment but is due for developments to be undertaken in this over the coming December month alongside other deliverables that will be evidenced in the final project report.

2.4 UI screens

Below are the initial UI screens, hand drawn and which formulate the main UI elements in game.

With the UI being a hand drawn element at this stage there is no evidence of transference into the grey box environment as yet, though this is set for instigation starting, approximately, in the last two weeks of December. (Subject to contingency due to the Christmas holiday period)

Below offers the reader a set of hand-drawn UI designs.

Start Screen UI

The start screen UI above shows the main user view of the game on start up. The UI mainmenu shall consist of a simple button to transfer to the "Area selection screen"

Area Selection Screen UI

The Area Selection Screen UI above shows the typical layout and functional elements of the on screen interface. The main body on the right hand side will hold a representative image that is used to denote the interests of that location. The option buttons to the left hand side offer the user succinct choices of exploratory destination. This screen will require a UI cursor element to assist the user in making a choice of destination.

In-Game UI

The above image demonstrates the main user interface in-game. The in-game interface will aim for minimal screen interference as much of the game-play relies on reading texts and listening to audio where applicable.

The sign identifying the Tor, or in the UI design for example; Jays Grave will be in the form of an icon (decisions over textual or image identifiers as icons are still under investigation due to the icons themselves being of less importance than the mechanics that will drive them, ergo icon mechanisms are the current concentration where the UI is concerned as they should prove a large part in the navigation of the game area)

The below in game (Grey box) image offers the reader insight as to how the game UI will appear during play.

2.5 Game Assets (Experimental and in-game placement models)

Please review the current grey box prototype that accompanies this first report for evidence of modelling assets and application of them into the Unity environment.

A full assets list will be delivered in the final report of this project, with preliminary assets delivered by the second instalment of this project report.

2.6 Prototype (Grey Box & Experimental Environment 1)

Typical image overview of the game level environment (representative not actual – due to no created terrain models for realism of environment at this stage)

3. Draft outline of the report.

This is likely to follow the development approach used – there is more information further on in the Guide as to report structure.

3.1 Use Cases

The below use case identifies the system, user, and developer along with their relationship to the system.

The use case below identifies user interactions with the system

The below use case identifies the general interactions from the user which move forward to create the set of requirements to be met by the system.

The below use case identifies the interaction between the user and game at the time of discovering a clue. This has been employed to demonstrate, at the high level, a mechanics requirement of the system by the user in order to deliver an in-game experience. The example is of finding a clue during exploration which signifies the path to exiting the area.

3.2 High Level Action Diagram

Explore the Tors of Dartmoor and obtain the clues to escape Action Sequence of events:

3.3 Activity Diagram

The following activity diagram expresses the nature of the high level Action diagram, but with semantics low enough to be used as a template to extrapolate code classes from.

3.4 Class Diagrams

The class diagrams are being completed at the time of this report alongside other documentation of the system and in concurrency to the project scheduled timeline.

However, due to research into the programming paradigm of Unity3D, it was found that the model did not conform to the strict OO paradigm one would normally aspire to follow. Instead, Unity3D is a component oriented environment whereby objects are created and self-contained (encapsulated) scripts applied to them, once this object is completed in the Unity engine it can be prefabricated and further objects of the same type can be applied as instances ensuring repetition of code does not occur due to the instance parent being the location of the instances origin.

What shall be available in following reports are Component model diagrams that identify object in the game that go forward to identify the system in diagrammatic form. Further research is under way into the nuances of correct component model diagramming techniques.

3.5 Sequence Diagrams

These diagrams will arrive for the second progress report of this project and will demonstrate key sequences that identify in game messaging between the users actions and the systems response

Part 2 deliverable report covering full research and

Gareth Wright 30076024

PROC303 –Personal Project

Acknowledgements

Abstract

Introduction

Acknowledgements 2

Abstract 3

Introduction 7

1 Problem Definition 8

1.1 Purpose 8

1.2 Focus group 9

1.2.1 Focus Group Members Background 9

1.2.2 Focus group conclusions 11

1.3 Project Scope 12

1.3.1 Game Feasibility study utilizing T.E.L.O.S 12

1.3.2 Outcomes 15

1.3.3 Deliverables 15

1.3.4 Roles/Responsibilities 16

1.3.5 Boundaries 16

1.3.6 Contingencies 17

1.3.7 Project Gantt Chart 18

2 Research 19

2.1 C# 19

2.1.1 C# “using” Directives 21

2.1.2 C# “using” Statements 22

2.1.3 C# Namespaces 23

2.1.4 C# syntax 23

2.1.5 C# structure 24

2.1.6 C# Common Influences 25

2.1.7 .Mono Framework 4.0 26

2.2 Unity 3D 27

2.2.1 Unity API & GUI 30

2.2.2 MonoDevelop 31

2.2.3 3d Modelling Tools 31

2.3 Other research 31

2.3.1 Game Vectors 31

2.3.2 Timing values 32

2.3.3 Game Components 32

2.3.4 Gamestates/ Level Management 32

2.3.5 Movie controls 32

2.3.5 Quaternions Vs Eulers 33

2.4.1 XP 34

2.4.2 SCRUM Research 36

2.5 Design Principles 37

2.5.1 Factory Design Pattern used with prefabs 37

2.5.2 Singleton Design Pattern used with objects 37

2.5.3 Other Design Patterns used.	37
3 Project design	38
3.1 UML Documentation	38
3.1.1 Package Requirements	38
3.1.2 Package diagram	38
3.1.2 Class Requirements	39
3.1.3 Class diagram	39
3.1.4 Class dependencies	39
3.1.5 Class diagram Rich	40
3.1.6 Class Dependencies diagram	41
3.1.7 Object model dependencies	41
3.1.8 Use case	42
3.1.9 Sequences	43
3.1.10 XP documentation	47
3.2 Game logic method explanations	47
3.3 Release Planning	47
3.3.1 Release Plan & User stories	47
3.4 Planned Iterations testing	48
3.5 Final Test Table	48
3.5.2 Final Developer Approval Tests	48
3.5.3 Final User Approval Tests	49
3.5 UI Design	50
3.5.1 Start Screen UI	50
3.5.2 In-Game UI	50
3.5.3 Game over UI	51
3.6 Elements of Graphical Design	52
3.6.1 Game Sequences (Unity Pro development)	52
3.6.2 Composite list of Scene Artwork and Models created	53
4 Fully Developed Game Code	53
4.1 Coded structure	53
4.1.1 Main Game Structure	53
4.2 Coded Base classes and Interfaces	53
4.2.1	53
4.2.2	53
4.2.3	53
4.2.4	53
4.2.4	53
4.3 Coded engine classes (if any???)	54
4.3.1	54
4.3.2	54
5 End User Documentation	54
5.1 Minimum Specifications	54
5.1.1 Windows PC	54
5.1.1 Android (TBC)	54
5.2 User Manual	54
5.2.1 Install Windows	54
5.2.2 Install Android ? (TBC)	54
6 Project Review	55
6.1 Critical Appraisal	55

6.2 Methodology Appraisal	56
7 Project Conclusions	56
7.2 Focus group Conclusion	56
7.2 What was achieved	56
7.3 Future developments	56
Appendix A	56
Appendix B	57
Appendix C	58
Appendix D	59
full code	59
Appendix E	60
Appendix F	61
Appendix G	62
Appendix H	66
Bibliography	68

Introduction

This project was composited for the reader to pick up and become more familiar to the nuances of C#, Unity 3D IDE game development and common aspects between the Mono & .NET Framework.

Component-oriented programming and design techniques that conform to the “Unity way” of code construction and other general developments have been used to bring a prototype development from an idea to fruition.

The project consists of a system build in the forerunning sections and that covers analysis of the system, research into the language and developmental aspects using C# and Unity 3D®, System design with some use of UML, conforming with agile development methodologies alongside elements commonly used in XP and Scrum practices.

The reader is invited to peruse the assignment and with the hope that what is covered within will increase a readers knowledgebase, without any or much prior experience of Component Oriented Programming, strongly typed language practices, design principles, and development using agile paradigms.

As such, this project assumes no prerequisite understanding of games development by the reader and attempts to allay any discrepancies with this idea using images and annotated explanations wheres possible.

There are several technical aspects found in this document, but care was taken to bring explanations to the high level using diagrammatic accompaniments where it seemed appropriate to example any complexities of the project build.

The purpose of this project was to deliver a game that can be played by all ages and abilities, using realistically textured graphics alongside simple control mechanics to aid with the provision of a fun gaming experience.

Further to this, the experience of visiting real areas of interest with comparable game locations mimicking the real areas geographics.

The deliverable was set to accomplish a fully demonstratable prototype game development based around the legends, facts and stories of Dartmoor. This iteration of the potential series focuses around the suicide victim Kitty Jay and the area around Houndtor, Dartmoor. This implied a supernatural element to the idea and this developed to be the premise behind this game`s theme.

This project has utilised the Unity 3D Engine, framework, and Mono version of the .Net framework to composite the iterative and final articles.

This project utilised the Mono framework, C# Language, and Unity3D® to allow more time to be given to code development and experiential learning and less time, with respect to time costly development areas of code and debugging such as garbage collection (a common predetermined feature in many newer OOP languages such as Java for example). There is also the considerably lengthy process of instigating an environment for the game to display and complex handlers for controlling the graphics hardware devices and user input throughout the game`s duration. These have all been contextually predefined when a project is undertaken using the Unity 3D® IDE and the C# Mono framework.

As a summary of the above, this project precludes to offer the reader a journey into the nuances of C# and Unity, 3D, some modelling and texturing and attempts to produce the below as deliverables

- Story led 3D scenario game idea, offering exploration, puzzle element, and textual information based on the local area and history
- Provide a wide demographic of players with an experience that is fun and engaging
- Rewarding story driven narrative
- Geographic replication of real Dartmoor environments
- Demonstrate a potential series led development idea through this prototype game.

1 Problem Definition

1.1 Progress Against Plan pt II

1.1.1 Project deviation summary

During this interim stage it was found that some aspects of the project needed readdressing to find a palpable solution to producing realistic environments for use in Unity 3D.

The original solution was to use STRM geographical data to produce a heightmap data set that could be incorporated with a plane, through displacement, inside a 3D modelling program.

This proved to be of some success and the following image demonstrates the initial results using this method of approach.

The below birdseye view is of the area Houndtor

The image, overleaf, has been offered to identify the heightmap data created from the USGS Google Earth topological data overlay plugin and imported into MICRODEM (a freeware DEM file software package)

The above image was the result of using the USGS plugin, which overlays heightbased colouring denoting darker colours showing lower regions and light colours representing the higher regions.

The above image helps to identify the issue raised in this progress against plan.

As can be seen, the visual data extracted from the above result did not present respective data that could be used to develop a detailed heightmap of Houndtor from, mainly due to the 25KM zoom in restraint that Google operates.

(Houndtor is the small green icon in the center of the map)

Further to this the resultant port into MICRODEM resulted in the below image which was also too vague to establish correct heightmap data from.

To resolve the issue a different approach was undertaken and the following image demonstrates the heightmapping result of using the original birdseye image, created by utilising the NVIDIA tools for normalising alongside techniques in Photoshop to create the normal map, then using that as the displacement data set to port into 3DS Max.

The above image and the image overleaf demonstrate the results of the displacement mapping techniques used on a plane visualised into 3D data.

The below image is used to demonstrate the first satisfactory result after being brought into Unity3D and converted to a terrain through script which takes object vertex data and applies it to a Unity Terrain.

The terrain reads the converted data as contextual heightmap values and applies this over a max spread of 65, 536 (effectively 64, 000) vertices.

Terrain Textures:

Rock:

the rock texture was taken from a real image, offset(using Photoshop) on both the x and y axes then, using other internal Photoshop tools the image was refined, clone stamped and blurred slightly to remove any seams in the image. This resulted in the below image which was formatted for tiling when painting the terrain in Unity3D.

Grass:

The grass image taken alongside the rock image reference, could not successfully be refined to produce a convincing grass texture. The consequential result was to completely invent the grass texture by using the "Render Clouds" option, under "Filters" tab in Photoshop. This is not an exact science, but using a combination of both Clouds, Difference Clouds, brightness, curves, Hue and saturation filters alongside the human eye, The below image was the result to be used in this second prototype iteration.

(NB: Further attempts to use a real location texture of the grass and moor bracken will be made and this is a temporary solution to suffice for the time being)

1.2 Focus group

The focus group has already been established due to the time constraints of the project.

They were continually utilized for their end user consultative input and inspirational direction:

The members of the focus group consist of ten people selected to bring diversity of character via age range and equalized numbers in terms of gender and are listed below as follows:

- Andrew Cuffe 36
- Dan Horsey 22
- Oliver Rosier 15
- James Wright 8
- Jacquie Webb 34
- Emily Jenkin 20
- Kayleigh Wright 25
- Jane Aisthorpe 39

1.2.1 Focus Group Members Background

- Andrew Cuffe 35 - Final Year Student (CGD), University of Plymouth

Andrew has an extensive experience and knowledge in, gaming since the age of 10 years old and represents the 31-35 age range in the focus group. Andrew is a knowledgeable person with some experience in the games industry.

- Dan Horsey 22 - Final Year Student, University of Plymouth

Dan is a keen games player and enjoys using his phone to find new games.

He is an accomplished programmer and has a good overall repertoire of gaming under his belt since early childhood. He represents the 21 – 25 age range in the focus group.

- Oliver Rosier 15 - Full Time GCSE Student, South Dartmoor Community College

Oliver is keen Xbox user and spends the predominant amount of his free time playing games. Oliver can bring opinion from the younger demographic of potential users of the game. He represents the 11 - 15 age range in the focus group.

- James Wright 8 – Primary school student, Elmswell Primary School

James loves all videogames and can bring a refreshing opinion over direction with respect to a potentially younger audience for the game

he represents the 8 - 14 age demographic in the focus group.

- Jacquie Webb 34 – Employed, South West Water

Jaquie is a keen mobile game player in her spare time and likes to play casually oriented games and could be helpful during the focus on the difficulty of challenge and speed of reward in order to keep the casual gamer interested. She represents the 31 - 35 age range in the focus group.

- Emily Jenkin 20 – 1st Year Nursing Student , Southampton University

Emily is a student in full time study and enjoys playing any game on any platform. As such she represents a good generalised level of potential user that holds peripheral interest in the games technical production but a good level of interest in it being fun to play. She represents the 16 - 20 age range in the focus group.

- Kayleigh Wright 25 - Graduate, Author

Kayleigh enjoys the limited time on casual games and also enjoys some time using console environments to play downloadable games on offer. She will be a great input with respect to feedback over quality of experience when obtaining the game through a download portal, such as the XBOX Live Marketplace. She represents the 21 - 25 age range in the focus group.

- Jane Aisthorpe 39 – Beauty Therapist, Self Employed

Jane is the potential wild card in the focus group as she has little experience with games, platforms or controlling them. But could be seen as a vital 'Devil's Advocate' to the group during discussions where suggestions may veer off into the technical realm, due to the calibre of some in the group, for example where user interfaces with complex control systems that require game playing experience to be a prerequisite to playing a game. She represents the 36 - 40 age range in the focus group.

1.2.2 Focus group conclusions

During the initial interview with the focus group a questionnaire was assembled to gain data regarding the types of game the focus group enjoyed playing. This questionnaire, alongside the collated results, is located at the back section of this assignment under the title Appendix B.

Focus group meetings were upheld throughout the project in the form of meetings where minutes of the discussion were recorded with those that could attend and the minutes forwarded to those who could not.

These meetings proved significant to the direction of the development during the design and development stage particularly. Alongside that, they group proved invaluable for the test phase.

In Appendix C of this assignment, one can find the meeting minutes that led to influences over the visual and logic design of the system.

1.3 Project Scope

1.3.1 Game Feasibility study utilizing T.E.L.O.S

A T.E.L.O.S study was used to establish whether aspects of the project and, indeed, the whole project is feasible enough to continue on to further subject research and design of the game.

This will go forward to compliment and populate the definition of requirements for the build.

1.3.1.1 Technical feasibility

By means of identifying the problems with their offset solutions, a technical analysis has been undertaken with a clear definition of requirements and there counter actions to create solutions:

Can the project be built by a single entity? Yes, The technology can be accessed by the project creator for all aspects of the build including System design, Coding, Art work, Testing, Implementations, and Marketing. Although some confabulations among peer groups for potentially cleaner and more succinct code practices may take place during development.

Is the technology available to implement the project successfully? i) C#

ii) Unity 3D Package

iii) Mono IDE

iv) .NET 4/ Mono Framework

v) Adobe cs6 suite

vi) Autodesk suite

vii) Blender

viii) Standardised UML modelling process

Is the necessary expertise and infrastructure available to carry out the task at hand? Not all the expertise will be pre-requisite to development. Some aspects of the project denote independent learning and research to take place before the project can move forward and onto the later code development stages.

These will be highlighted throughout the projects chapters where a necessary basis for understanding was developed before moving forward.

Will the project meet expectations regarding it users.

The project will remain fairly subjective by means of a focus group, with respect to meeting expectations as the finished product is a game and will require feedback at the final release stages to establish if expectations were met, although the focus group will also be key during development to ensure expectations are en route to being met.

Are there any portability issues? There are no real issue for portability, aside from Unity3D requiring a pro license to actually release a full version of the development.

10 different platforms are already supported in Unity3D for cross platform portability

1.3.1.2 Economic feasibility

This section covers the questions relating to whether the project will hold some level of return, perhaps through remuneration, from the developed outcome of the project.

With respect to this, the finished game, which represents the outcome and goal of this project shall be marketed as a free to download software application/ game and will not extensively focus on monetary remuneration, although this may be developed over the project's duration as an extended idea with respect to future developments and any updates for the product.

A cost benefit analysis on this project would be better analysed by a feasibility study over whether an end product can be obtained against the completion timeframe– this can be demonstrated through viewing the Gantt chart , which identifies critical timescales for completion during the various stages in development.

However, there are items that must be obtained so that a final product may reach an audience outside of the focus group. These are laid out as follows:

- Unity Pro License*, allowing official deployments.

*As an aside, When obtaining Unity License it does not give you a license to deliver over all platforms, namely IOS and Android will add further costs to a developer hoping to reach these audiences. Critically this should not be seen as a negative as with IOS, for example: users of IOS often pay for their apps when they appear free on Android.

1.3.1.3 Legal feasibility

Legally there is little, by way of concern, due to the game being developed as an original concept.

In real terms this could be argued through the fact that, although there are multiple types of games available, free or otherwise, the game type has no real definition with respect to the legal implications of naming a genre, for example "Platformer" or "Arcade".

However, there is concern to be raised with respect to game title and likeness, in that it should not be identified with exactly replicated traits or elements of any game currently available, simplified this means one must certainly not pass off other artists work as original and must create the pertinent acknowledgements towards any such outsourced components, if the game includes anything by them. As is the same with source code, correct acknowledgements must be given and licensing restrictions must be adhered to, whatever the individual stipulations may be.

With respect to the above paragraph in terms of addressing original code and artwork, references will be made to authors throughout the project, where necessary with respect to licensing restrictions attached to that code. It is important to mention here that, aside from the predetermined functions laid out upon IDE project creation in the Visual Studio 2010 environment, all code will be bespoke and will not be taken from any code repositories via copy and paste techniques nor shall any artwork be used that has not been originally created due to one of the main focuses on the entire project is to become proficient in the application of code to produce a working product. As such, only real understanding can come from self development of a working knowledgebase of the codes structure. Experientially, this has been found to be better obtained through practice rather than through pasting other authors source code only to find it can take as long as writing it originally to iron out the problems that arise from implementation it into one's own code.

1.3.1.4 Organizational feasibility

With respect to organizational feasibility, this project is not undertaken for an organization or pertinent business client to adopt into any current system.

There will be scope for further research into this aspect of feasibility at a later opportunity in the project when the final prototype is released for scrutinization on the web, post project hand in date.

This system is not a system that is built to solve a businesses problem but was more designed to present a novel method of educational digestion through interesting exploration game-play. The problem in hand could be surmised, at best, to be that education does not reach all learners equally and that opportunity to deliver pertinent history driven content themed for engagement and enjoyment may also help to improve the retention of information by familiarisation to the experience along the way.

Ultimately, the game may or may not prove to deliver this element and will remain subjective until user testing has started.

1.3.1.5 Schedule feasibility

The project schedule was found to have remained feasible and realistic against completion of the project overall. At the time of writing, the schedule has continued to demand time but has also offered moments of significant progress, with respect to developing a grey box in a matter of weeks for the game idea, whereas this had been scheduled to for late December at the earliest, alongside class structures in place for adding code to the next iteration of prototype. These have currently been superseded by the quick fashion in which Unity offers usable results.

In order to control the timeline of development the project has utilised A Gantt chart to track phase deadlines. These phases will be broken down into their composite parts and they have been tracked using the Gantt chart.

Critical path analysis was not used in this project as the path is being undertaken by a single entity and the Gantt chart has proven to be a sufficient indicator of time constraints and with the addition of utilising Agile practices for the developmental stages there will be a time slot where aspects of the project can be added.

1.3.1.6 Telos Conclusions and Recommendations

The game is feasible for development as there is no actual real world client that is relying upon a working model to assimilate into their business in some way, although it must be emphasized that the project should hold equal weight to that of a corporate development by means of it focus group being the client who will ultimately critique it for quality and sufficiency.

1.3.2 Outcomes

- Finalized game with future development potential
- Better familiarization with techniques and documentation tools using the UML 2.0 specification
- A time served understanding of development processes commonly found in games

1.3.3 Deliverables

Below is the list of self-perpetuated deliverables the project has been designed to provide:

- System documentation via UML (Use cases, sequence diagrams etc)
- A physical game available to play over multiple platforms but predominantly demonstrated on Windows and MAC OS:

Historical in-game information regarding the local area

Research into C#, Unity 3D, .Net/Mono Framework 4 component library, Comparative elements to influential languages such as C++ and Java, and pertinent definitions with general research given to types commonly used in game Unity developments, including the use of component classes to conform to the pillars of Object Oriented Code and Design, with respect to the Unity 3D® Component Oriented way.

Next is the finalized list of required deliverables to complete the system as laid out after progressive discussions with the focus group.

- Challenging 3 Dimensional investigation/exploration game
- Visually appealing graphical environments based on real landscapes and areas of Dartmoor
- challenge and reward elements
- intuitive game control system and UI navigation
- Free to download (with the possible opportunity to extend ad support to see revenue/ return on time invested if potential found)

1.3.4 Roles/Responsibilities

The roles and responsibilities are split into internal and external sections, and have been laid out below for the reader to view.

Internal Roles Responsibility

Gareth Wright (Project Manager) Deliver finished project

Gareth Wright (Analyst) Deliver analysis of requirements

Gareth Wright (Designer) Deliver system design

Gareth Wright (Models & Artwork) Deliver game artwork

Gareth Wright (Audio) Deliver game audio and sound effects

Gareth Wright (Developer) Deliver small test passed game components

Gareth Wright (Tester) Deliver testing feedback

External Roles

Focus group members (User Requirements) Deliver a final requirement from the system

Focus group members (Artwork & Model critique) Feedback pro's and con's of system design

Focus group members (Audio critique)

1.3.5 Boundaries

The boundaries, laid out below in a bullet list for the reader to identify them, are as follows:

- At the end of development the game will have only been ported to Windows and MAC based OS `s alongside the Unity Webplayer, but with Unity`s safety net of cross platforms the scope to deliver is restricted only to the likes of Blackberry`s RIM and insufficiently specced machines running the array of Unity supported platforms.
- The project does not consider level editing for the user in the game
- The project development does not boast a professionally comparable game but rather a prototype dfeivering a proof of concept
- The project is based predominantly on the experiential gain of coding in a new environment and language using a deliverable product (a game) to maintain productive focus and as such will require further , post degree, to ensure conformances are met to deliver this to potential investors, if possible.

1.3.6 Contingencies

There may be call for a reduction of game dynamics in order to complete the project on time.

The ideas tend to flow well when speaking in context of the possibilities one might achieve when creating a game and can grow to become an elaborate, potentially unfeasible, project with numerous bells, whistles, smoke, and mirrors as part of the finished product.

It is the bells and whistles areas that will unfortunately suffer in the development timeframe leading to time saving reductions in the entire projects scope at the latter end, if required. This contingency plan will take advantage of areas that are seated in the refinement phase of developmental stages with respect to number of available game locations, task linearity, storyline, AI (Artificial Intelligence) thusly making a reduction in additional time spent on game logic in the process.

These aspects of the prototype must be embodied at the very beginning of the development and, as such, the game should not overly suffer with respect to its functionality or over the documentation of the project as these features must be present in order to provide a good foundation on which to lay a game premise.

As for the planning of a contingency strategy, this is clearly marked in the Gantt chart denoting the overall project time scale. There will be marked points, distinguishable in black, showing days clawed back through early completion that can be utilised in areas that may have fallen behind schedule. The contingency allots one day to review current status prior to a point of no return, so to speak, to establish if the contingency needs to be put into action or ,overtly, if any lagging areas in development can be recuperated and retargeted, with respect to recuperated days during the earlier research and analysis stages.

1.3.7 Project Gantt Chart

The below diagram offers the timeline overview of the project according to the Gantt chart data recorded during the project lifetime.

Both Gantt & Tracking Gantt charts pertaining to this timeline can be located at the rear of this assignment under Appendix F.

2 Research

2.1 C#

C# is a modern managed programming language that can be used to provide a type-safe (accesses only memory locations that it is authorized to)

The language has a number of common uses and, though not exhaustive by any means, have been listed below for the reader.

- Traditional Windows client applications (including Business and Games)
- XML Web services
- Distributed components
- Client-server applications
- Database applications

The language is described as ‘...highly expressive, yet it is also simple and easy to learn.’ (Microsoft, 2012), and as such this section of the research has endeavored to uncover the fundamental core knowledge required to produce an output from a C# program.

With this in mind, we can denote C# to be a multi-paradigm language that is designed to be object oriented in nature and strongly typed meaning restriction is placed upon type operations and further denotes how operations handle the intermixing of different data types.

Below follows a non-exhaustive list used to identify synoptically based information about the language.

Designed & developed by:

Microsoft corporation™

Born:

2001 (current version is 4, beta version 5 released on February 29th 2012 alongside Visual Studio 11 Beta™)

Multi-paradigm:

- Object Oriented
 - Structured
 - Imperative
 - event driven
 - functional
 - generic
 - reflective
- Typing disciplines:
- Static
 - Dynamic
 - Strong
 - Type-safe
 - Nominative
 - Partially inferred

Platform 1:

Common language Infrastructure (CLI)

Platform 2:

Common Intermediate Language (CIL)

Runtime:

Common Language Runtime

(CLR) = Executable code and runtime environment core of the Microsoft .NET framework.

= second platform neutral language that maintains compatibility to .NET

= Compiles CIL to machine readable code (10010010)

Influential languages:

- C++
- Java
- Eiffel
- Modula-3
- Object Pascal Licenses:

CLR is proprietary for Windows.

A cross-platform compiler known as Mono has a dual GPLv3 and MIT/X11 license with the libraries conforming to LGPLv2, Dot GNU meaning GPL & LGPLv2 together.

These licenses are beyond the scope of the project and as such will not be discussed further but links to their respective specifications can be found in the bibliography section of this document.

The information above can be found in an official capacity at the Microsoft website entitled C# Language Specification (Microsoft, 2012) and as previously stated, this list is not exhaustive. For the latest information regarding the specification, one can find the URL located in the Works cited section at the back of this assignment.

2.1.1 C# “using” Directives

Statements, which can be found at the very top of any C#-class are known as “using” directives.

The code diagram below should help to assist the reader in identification of these in any C# class:

A using directive has two main uses and these are briefly discussed below.

1. Allowance of type qualification through use of its namespace allowing shorter access code.
 - a In the above screenshot, one can ascertain there to be a using directive stating “using System.Text;”, this using directive allows one to code a simple output to the console thusly;

C#

```
Console.WriteLine("Hello");
```

Unity 3D alternative

```
Debug.Log ("Hello");
```

By removing the “System.Text” directive one would be forced to use

System.Console.WriteLine("Hello"); Instantly lengthening the affair of producing a simple output statement. Factor this out to larger scoped programs, for example; a game, and one could agree the use of directives to bring namespaces into scope that contain useful reduction practices to produce code exemplars the C# language as a tool for spending more time coding and less time typing.

2. Alias creation for namespaces and types, known as a “using alias directive”

The below code snippet, taken from the Microsoft MSDN website on C#, has been used to example the use of alias directives inside the languages structure.

namespace PC

```
{
    // Define an alias for the nested namespace.
    using Project = PC.MyCompany.Project;
    class A
    {
        void M()
        {
            // Use the alias
```

```

        Project.MyClass mc = new Project.MyClass();
    }
}
namespace MyCompany
{
    namespace Project
    {
        public class MyClass { }
    }
}

```

(Microsoft, 2012)

As can be seen, the alias directive can be delivered internally but. This was found to produce no effect that which could result in a conclusion over whether accessing a namespace or type through an alias made any real difference in opposition to use 1, and this is concurred on the website Using alias directives, (Microsoft, 2012).

2.1.2 C# “using” Statements

The using statement is created to obtain one or multiple resources whereby the statement is executed and consequently ensures the resource is disposed of. The resource (using statement) is composited of a class or struct designed to utilise the .NET framework System directive known as “IDisposable” which is part of the managed code paradigm behind the C# programming environment and ultimately Unity development with C# and Mono. It uses a method “.Dispose()” to return indication that the resource used via access through the using statement is required no further.

As such, the using statement can be viewed as read only, in terms of importation. It is also broken into a three part process defined by an acquisition, a usage and a disposal stage.

Furthermore, an important aspect to consider when undertaking the creation of a using statement, is the try clause which demands a usage to be implicitly enclosed, and the finally clause that disposes a resource, as without one may find no exception to be thrown thusly nullifying the call to “.Dispose()”.

Unity operates the using namespace statements and it is requirement to add to create working code in Unity.
using UnityEngine;

2.1.3 C# Namespaces

Namespaces can be viewed as a tool whereby various types can be correctly organized.

A conceptualization may allow an easier insight into what a namespace represents. For this the reader is invited to imagine a folder structure, like that which can easily be seen in any wimp interface (for example; Windows or OSX). These folders allow files to be saved within them and reference is made to their storage location through them. This in turn allows one to store an identical file with an identical name in two different folders with different folder titles. This is the same concept for namespaces. In this context, each namespace (folder) allow classes of the same name to be accessible through its namespace identifier and in turn fully qualifying it as unique.

A diagram below has been created to assist the reader in visualization, with respect to the above discussion.

As can be seen in the above diagram, one can utilize the Namespace structure to access classes, interfaces, and methods of the same name and type allowing for code reusability in the creation of assembly dll's for example, and relying more on the scope convention accessed via a top down approach which instantly avoids naming conflicts. Generally namespaces contain classes, structs, interfaces, enumerations, delegates and can also be found to contain other namespaces.

2.1.4 C# syntax

In C#, statements are created to perform actions and composite these statements into methods (a common artifact of Object Oriented Programming, and can be found later discussed under the heading Common Influences). A very common method found in C# is the Main(String[] args) method which acts as a common initial interface that is read by the program class that filters through what is known as the Content Pipeline.

2.1.5 C# structure

The structure of C# has been predominantly influenced by its forfather languages, namely C++ and Java (later discussed under the sub section heading: C# Common Influences). It takes the building of classes to create one or more files that composite a program`s code makeup and can cater for use of no or multiple namespaces containing the common types identified in the above discussion regarding namespaces.

As such, the below code should adequately assist the reader in identification of the C# code structure.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace TheLastHuman
{
    //class
    class MyResearch
    {
    }
    //struct
    struct ResearchStruct
    {
    }
    //interface
    interface IReasearch
    {
    }
    delegate int myResearchDelegate();
    //nested namespace
    namespace NestedResarchNameSpace
    {
        //nested struct
        struct NestedNamespaceStruct
        {
        }
    }
    //main program class
    class ResearchMainClass
    {
        static void main(string[] args)
        {
            //program code executed here
```

```

    }
}
}

```

2.1.6 C# Common Influences

Instantly upon starting to write code with C# one can ascertain its influences to have predominantly developed from C/C++ and Java

A fair dichotomy would be to state that it is loosely based on C/C++ and similar to Java, with respect to being a language that uses a common intermediary platform to pipeline source code into machine readable language. With respect to C/C++ the C# language does not have an exclusive class library that it can call its own but, instead makes heavy use of the .NET Framework which is a comprehensive class library in its own right, and is not exclusive to the C# language alone.

C# types are similar to C++, with respect to the available range of basic types, but where C++ relies on the platform to denote the length of the basic types, C# resolves a fixed length to each type, for example; a C++ long type depends wholly on the platform bit type meaning a long could be 32 or 64 bits in length depending on the platform architecture, whereas C# equates a long type to 64-bits and negates whether the platform architecture is 32 or 64-bit. Classes and structs can be seen as practically the same thing in C++, with respect to them being reference types (due mainly to its influence from C), whereas C# tends to separate the two quite distinctly, for instance a C# struct cannot contain initializations of a field. This is better explained through use of a code diagram.

As C# does not allow the initialization of fields in a struct, the following code would be invalid and an exception would be thrown at compile time.

```

struct AStruct
{
    public int x = 20;
    public int y = 20;
}

```

The below code shows the correct method by which to declare, instantiate, and initialization of a struct in C#

```

struct AStruct
{
    public int x;
    public int y;
}

```

Inside public static void main().....

```

AStruct myStruct = new AStruct();
myStruct.x = 20;
myStruct.y = 20;

```

What can be concluded from the above code examples is that C# differentiates structs from classes by type definition thusly.

- C# class = reference type
- C# struct = value type

This simply equates to meaning structs can be objects that conform to behavior similar to built-in types like double, int, long etc.

So to conclude this difference C++ passes reference to a struct when it is used in some method, the same as C++ classes and C# passes a literal copy of the struct when used in some method, and leaves referencing to be used for classes. This is achieved by the change of approach when its new operator is called, by allocating to the heap and

waiting for instantiation to create space for it on the stack. C# yields performance gains due to this as one can work directly with the struct through its instance instead of by reference to it meaning one can pass actual values instead of a reference to them.

Java is another predominant influence to C# by means of its similarity in structural and OO coding paradigms, the fact it is also compiled to an intermediary language prior to machine executable code is also complimentary to it but. One aspect that has influenced C# through Java more than any other language is C# adoption of Unicode character encoding for char and string types that then allow for complex scripts like Chinese, Arabic or Japanese to display correctly in a written program and overall can be seen to resolute multiple internationalization issues and current systems are designed to run programs faster under Unicode enabled software.

Effectively C# can be viewed upon as Java, but with a different set of Component libraries to use(.NET or Mono).

2.1.7 .Mono Framework

The Mono .NET Framework is the latest stable release of a framework that houses the class libraries that C# relies upon to effectively interoperate between it's classes.

As a comparator, below is a description, found on the Microsoft website and which moves forward to establish a synopsis that describes what the .NET framework actually is...

The .NET Framework is an application development platform that provides services for building, deploying, and running desktop, web, and phone applications and web services. It consists of two major components: the common language runtime (CLR), which provides memory management and other system services, and an extensive class library, which includes tested, reusable code for all major areas of application development.

(Microsoft, 2012)

With the above in mind, we can use the description found on the Mono website to see where the differences lie...

Mono is a software platform designed to allow developers to easily create cross platform applications. It is an open source implementation of Microsoft's .Net Framework based on the ECMA standards for C# and the Common Language Runtime. We feel that by embracing a successful, standardized software platform, we can lower the barriers to producing great applications for Linux.

(Mono, no date)

Analysis of the above two statements shows that Mono utilizes the same .Net framework as Microsoft, but have a distinct intention to use it as a standard across platforms outside of the Microsoft only compliment., particularly with Linux in mind.

As such, the .NET and Mono framework should commonly be viewed as transparent to developers using it. For example, if a piece of software were to be installed the executable may require some .NET/Mono framework library and install that to one's system and one need never know why or ever use it. However, with developers utilizing the .NET/ Mono Framework, one can view it as a vital inclusion to successfully and safely building projects from code in C#.

Explanations of provisions that the framework offers to programmers can be found on the Mono website pertaining to the Mono .NET Framework but, below is a round up what it can be seen to provide.

- Memory management
- Common type system (no longer relies upon compiler definition of a type)
- Extensive class library that includes allowances for programmers to handle low level operations safely
- Development frameworks and technologies designed for specific areas like networking, web pages, multimedia applications and more.
- Interoperability of language due to intermediary code produced by the CIL (Common Intermediate Language) then compiled at runtime by the CLR (Common Language Runtime)
- Retro compatibility for earlier .NET implementations, which should, mostly, not require recoding
- Cross platform coding

2.2 Unity 3D

Unity 3D (current version at time of writing 4.0.1) is a complete game engine system that allows novice and experienced developers to create speedily prototyped developments related to gaming and GUI programming. It couples a cross-platform development tool that includes a GUI and code driven IDE alongside the ability for full integration with third party or self created external developer libraries.

The product was born from a game development tool originally created for Apple's MAC OS X circa 2005 and has continued to evolve to now cater for development across 10 separate platforms making it a truly cross platform tool. There are two versions of Unity 3D® available: Pro and Community Edition, of which the differences are further discussed at a later part of this sub section.

The below establishes some of the new features provided by the fourth iteration of Unity 3D, though no distinction is made between pro and Community at this juncture.

- Mecanim

Unity 4's upgrade from version 3.5 brought with it the release of the Mecanim Animation System, which greatly widened access to believable and real world animations that can be utilised inside of Unity® and couples to an improved shuriken world collision system making the potential for AAA like animations possible to all (with the caveat of relative experience).

The following extract is from the Official Unity website and aimed to offer the reader a quick peripheral understanding of the Mecanim animation system...

Mecanim, Unity's uniquely powerful and flexible animation system, brings your human and non-human characters to life with incredibly natural and fluid motion.

Mecanim is a complete turnkey solution for animation in games. It abolishes the need to spend expensive development efforts to integrate 3rd party middleware. Mecanim is natively integrated with and optimized to run in the Unity Engine. From the Editor, you get all the tools and workflows needed to create and build muscle clips, blend trees, state machines and controllers directly in Unity.

Animate even vast armies with retargeting in Mecanim. Unity's stability and power, combined with new optimizations, such as skinned mesh instancing, ensure smooth runtime performance.

(Unity Technologies, 2013)

- Real time shadows introduced in Unity 4

Objects can cast shadows onto each other and onto parts of themselves ("self shadowing") (Pro Only)

- Flash and Linux native authoring

Publish directly to flash or Linux OS

- Cross Platform Dynamic Fonts

A single True-Type font is all that is needed to work across Unity's platform delivery spectrum

- DirectX 11 support

Deferred lighting and shader model 5 alongside parallel processing (GPU processing)

HDR images

Linear space gamma correction

These edition have some distinct separations, with respect to performance and access to some of the Unity Engine's capabilities mentioned above, though these differences by no means hinder the development of a game's proof of concept, with respect to prototyping the functional, controllable, and general graphical aspects.

The screenshot below has been used to assist the reader with visual identification of the general Unity (Community Edition) UI:

The following table overleaf was laid out to identify the main differences between Community and Pro editions of Unity 3D.

The table is not exhaustive as such, but offers clarified differentiations between the two editions, and identifies areas where Unity 3D Community Edition cannot be utilised to achieve some desired effect. This further assisted to denote

potential consideration for the game in it's early development stages prior to any move in to Unity Pro at the final stage iterations.

Is Supported? Community Pro

Supported platforms PC with DirectX 11 shader model 2.0 PC with DirectX 11 and supporting shader model 2.0 upwards

Shader Model 2.0 (Phone has no support for custom shaders) 3.0 and upwards

(Unity Pro supports applying custom shader extensions)

Physics NVIDIA PhysX® NVIDIA PhysX®

Particle System Shuriken Shuriken

Max texture Size 1024 4096

Max Cubemap size 1024 4096

3D Texture support Not Supported

Browser Integration

Deployment Windows, Linux, MacOS, Android, IOS, Webplayer, Flash, PS3, XBOX, WiiU Windows, Linux, MacOS, Android, IOS, Webplayer, Flash, PS3, XBOX, WiiU

Realtime Shadows Not Supported

HDR Not Supported

Light Probes Not Supported

Optimized Graphics Directx & OpenGL Directx & OpenGL

ShadersBuilt in & Customisable Built in & Customisable

Lightmapping Generated or imported Generated or imported

Global Illumination Not Supported Beast lighting system extra

Dynamic Batching

Static Binding Not Supported

Terrains

Full Screen PPE (Post Processing) Not Supported

Occlusion Culling Not Supported Umbra

Deffered Rendering Not Supported

IK Rigs Not Supported Full IK positioning using Unity rigs

Mechanim Sync and Curve Not Supported

Navmeshes Not Supported Navigation baking and high performance path finding & crowd simulations

Dynamic Obstacles, Dynamic Priorities

LOD (level of Detail) Support Not Supported LOD grouping for optimization over distance

Audio Filter Not Supported FMOD filters

Video Playback & Streaming Not Supported

2.2.1Unity API & GUI

The Unity Engine is an Application Programming Interface based upon the solid, standards driven, .NET Framework and contains methods that are commonly created in game development. An epitome of this would identify itself in something like the loop scenario, which a developer must usually template prior to building a game. This loop moves forward to become an essential part of any game by making up what is known as the game loop.

The game loop in Unitycan be accessed in the Update() function which runs every game frame, or the FixedUpdate() function which runs every physics based game frame

In terms of development using pure C#, one would find the need to import C functions that could take advantage of low level speed of access in order to create a game loop useful enough that it could update in time to the speed of occurrences provided already in Unity's Engine. This would be carried out through accessing timing information by importing a c struct into C# and using it to access changing values and pass them to C# to make a fast loop. Below are

the two methods automatically generated upon the creation of the project solution and which allows for the programmer to instantly start coding into a timed updatable loop.

This is just a part of the game loop Unity has and below is a quick run down of the execution order of this loop, as laid out by Unity Technologies.

Load First Scene use ...

- Awake(Called prior to start methods, but post prefab instantiation – cannot be called until a game object is made active or a scripted function is executed on a game object)
- OnEnable(called on active objects to instantiate them prior to loading a scene/level)

Before the first frame update use...

- Start(If enabled this is called before first frame update)

In between frames use...

- OnApplicationPause(Called at the end of a frame, but dependant on a game pause action being executed first)

Update use...

- FixedUpdate(Can be called multiple times per frame, allows physics calculations to be updated immediately)
- Update(Main Update loop called once per frame)
- LateUpdate(Called once per frame after Update has completed)

2.2.2 MonoDevelop

The MonoDevelop IDE is an open source code editor that can be used to develop code alongside Unity 3D, it is bundled in the Unity 3D download as the editor of choice for Unity Technologies.

The development editor also provides hinting and code completion steps, similar to the likes of Microsoft Visual Studio's intellisense™, when writing code.

Key to MonoDevelop's success, with respect to Unity, is the integration of Unity's debugger to deal with any incorrect coding issues

The Ultimate edition of Visual Studio 2010™ provides the developer with tools to view the system build's architecture and create class, object, method, namespace, and other dependencies diagrams.

The IDE is optimized for agile, iterative approach to program developments and comes with many extra tools to assist with sticking to time boxed and other sensitive deadlines, and some of these will be recorded in the appendices section of this report relating to the later section and which records the game development project.

2.2.3 3d Modelling Tools

The following modelling tools are under continual evaluation to acquire the basic skills required to operate them effectively.

The below listed programs were found to require in-depth practices and understanding to achieve any respectable modelling results.

Autodesk:

- 3DS Max 2013 – This software covers all aspects of 3D model creation, but has the highest complexity due to how much is incorporated into one all encompassing program.
- Maya 2013 – This software is the game development refined version of its bigger bolder sibling 3DS Max. It is used industry wide to create professional and optimized character and environment models.
- MudBox 2013 – This tool is for the initiated user and offers some very important but basic elements required to build a 3D model including a 3D paint aspect where model painting in the application provides automatic UV mapping on export simplifying one of the early hurdles met in 3D model development for the prototype development.
- MotionBuilder 2013 – This tool offers the user the ability, alongside other plugins like Brekel for Kinect™ to attempt rudimentary motion capture "MoCap". Brekel is an open source implementation that accesses the Kinect

hardware and utilises data from the camera and sensors to record the positional, rotational, and translational data from a calibration to a users actions.

2.2.4 Bitmap Tools

The following bitmap tools were utilised to establish certain file types and to manipulate image data to the required standard before applying them to any game elements requiring texturing.

Adobe:

- Photoshop CS6
- NVIDIA texture tools plugin for Photoshop (normal, bumpmapping)
- Alien Skin Eye Candy plugin for Photoshop (additional filter plugin to enhance images)

2.3 Other research

2.3.1 Game Vectors

All games use math of some degree or another and this project was found to be no different.

This was required so control could be established over game AI that could use the positional data to hunt for the player object during game time.

As such, it was found that Vector3, a public struct located in the Unity framework is predominantly used in game development to establish information regarding the x, y, & z coordinate positions at any time over a games loop, and could be retrieved to provide data useful for establishing positions and directions.

The Vector3 has common operations that stretch to other classes like the Quaternion or Matrix4X4, which are useful in Unity and general game development to rotate or transform vectors and points in 3D space.

Below is a code diagram showing the Vector 3 class declaration and initialization.

Declaration:

```
Vector3 myVector;
```

Initialization:

```
myVector = new Vector3(1, 1, 1);
```

Rectangles are also a type in XNA that can be used to hold positional data that reflects the outline shape of a size specified quadrilateral. These offer a developer another way by which to lock on to objects on the screen, by passing the object position to the rectangle as a parameter, and have them affected by manipulation of their respective bordering rectangle or used to detect collision of one rectangle and another, or indeed one rectangle and a vector2

Global declaration:

```
Rectangle myRect;
```

```
Int x = 50;
```

```
int y = 50;
```

```
int width = 512;
```

```
int height = 512;
```

Initialization:

```
myRect = new Rectangle(x, y, width, height);
```

2.3.2 Timing values

Update

Fixed Update

Time.time

Time.deltaTime

2.3.3 Game Components

GameObject

2.3.4 Gamestates/ Level Management

The Unity Engine offers Gamestate management through its GUI based “Build Settings”, found in the “File” menu of the main program’s window (PC).

This gamestate management can be used to associate level numbers with scenes in the game. The levels can be utilised via code to effect a level change during a key element in the game like completion of the current level (the game examples different levels to explore with different objects to obtain).

2.3.5 Movie controls

Currently there are no movie elements in the prototype due to Unity 3D community edition not offering indie developers the ability to access the MovieTexture method.

Below offers the reader the method that will be invoked to

It is with the below line that the movie will be invoked during the methods execution....

```
renderer.material.mainTexture.Play();
```

....Inside a regular Update() method call.....

```
void Update ()
{
    if (Input.GetButtonDown ("p"))
    {
        if (renderer.material.mainTexture.isPlaying)
        {
            renderer.material.mainTexture.Pause();
        }
        else
        {
            renderer.material.mainTexture.Play();
        }
    }
}
```

The above has been laid out to demonstrate the code that is intended for use once the last iteration period of the development is reached and the Unity project is transported to Unity Pro. It is here that full advantage will be taken of the MovieTexture calls to apply movie sequences to the experience.

2.3.5 Quaternions Vs Eulers

The key behind Quaternions lie within complex number theory, represented as $a + bi$ mathematically (with the “i” commonly found as “j” in the electronics/computing world). The “i” represents the number as being imaginary or figmented, but also representational of a number line that spans 2 dimensions. This, in short, allows one to solve cubic equations mathematically and is especially handy for one to establish the position of a cubic element in a three dimensional game world then calculate a rotation or interpolation related to its position.

In game development, and specific to Unity 3D Quaternions are used because they are fast to calculate leading to better optimized code that alternatively might use Euler Angles, a potentially more human readable approach to Quaternions.

Most notably, Quaternions are also favoured due to their ability to avoid Gimbal lock – the loss of one degree of freedom in 3D space caused by two of the three(3D) available gimbals (an imaginary ring of rotation around 3 axes) fall into parallel, forcing a rotation to lock into a degraded 2D space rotation rather than the required 3D.

Using Euler Angles this issue is counteracted by the addition of a fourth gimbal (additional outer ring) and continually angled at a 90 degree offset alignment to the centralised gimbal. Without this additional fourth gimbal Euler Angles present themselves as flawed to correctly define rotations in 3D space, but can be found as important in the definition process of a game coordinate system in a 3D space.

Why are Quaternions important and correct for 3 dimensional space rotations?

Mathematically, they describe a rotation* in a single pass by specification of the angle in radians and the normalized axis of its vector to turn on. This is 2 times less complex than the Euler solution, which requires three successive descriptions based on each gimbal to make the same rotation.

*A Quaternion rotation is stored mathematically as a tuple (ordered list of elements) and computationally these are stored as Matrices, Vectors or Arrays.

(In terms of computing and Unity, Quaternions can be used to operate on a bunch of 4X4 matrix/vector coordinate data values describing each “gimbal’s” positional “x, y, z, & w” by complex number, with “x, y and z” denoting the imaginary position and “w” denoting the real (normalized) number part)

2.4 Further research on Agile practices and methodology concerning XP and Scrum

The practice of iterative development has been touched upon during the second year of the foundation degree and as such one chose to use a combination of XP and Scrum to document the system.

Typically, these agile practices conform to iterative development and incite concentration to predominate the development and test stages, whereby a deliverable small component of the game can be released per time boxed sprint.

XP conforms to practices used with TDD (Test Driven Development), whereby developers write components to pass tests and, once passed, is released to involved users (focus/feedback group or “white boxers”) for use and anything missed, though the development process should catch any issues prior to their receipt of any completed game component through the agile practice’s efficacy ingrained through its process.

2.4.1 XP

XP is an agile method used to develop systems that concentrate on the documentation being encapsulated in the code. This is predominantly for the method’s key concentrator, a practice known as pair programming.

Pair programming was not undertaken as the project was a self motivated and self reliant exercise, meaning no other developer party was involved during the process at any stage. Though this is not sound practice under the XP banner, the practice of commenting the code to read the design was still put into practice to conform with the agile paradigm’s specification.

So, Whats in XP’s engine?

Research has led to the conclusion below, which is not a definitive explanation but sufficient enough that one may understand the practice and involvement a little clearer.

Firstly, we must establish the classic steps in the XP process:

- Requirements Analysis
- Specification
- Design and Architecture

- Coding
- Testing
- Documentation
- Maintenance

The above points move forward to create the iterative systems development life cycle in XP. Secondly, we must establish how it differs from high ceremony (heavily documented) methodologies.

- All requirements will not be known at the beginning
- Requirements will change
- Use tools to accommodate change as a natural process
- Do the simplest thing that could possibly work and refactor mercilessly
- Emphasize values and principles rather than the process (high level abstraction)

<http://www.xprogramming.com/xpmag/whatisxp.htm>

The above image reflects the practices of XP and makes clear the paradigm it offers, with focuses aggregating in the design and development aspects of the method, as opposed to some methodology like the waterfall method that produces great time costs at the analysis and design stage prior to building anything of worth to a system and reliance that the design was solid enough to pass through to practical code implementation without any issues. The last part of the previous statement rarely occurs flawlessly, and the waterfall method entertains no possibility of design and changes after the design stage is finalized.

2.4.2 SCRUM Research

Scrum is used as a metaphor for to coincide with the same premise of the rugby scrum. When a scrum is formed it becomes an short but intense time span where clear, succinct and speedy interaction between the team players (development team) and getting the rugby ball (product deliverable) out to their team's fly half (Client).

The metaphor stretches even further to denote the level of interaction the fly half must have with the scrum to understand where the ball is and instruct of changes that the scrum need to make to effect the ball enough that he can accept it. Even further, we can denote the scrum half in the middle of the scrum to be the project team leader who must battle hard to ensure that the ball is delivered through his team.

As the above metaphorical explanation of scrum is hoped to incite, Scrum is a quick deliverable, time sensitive based style of handling development stages. The emphasis is to get the product deliverable out as fast as possible by continual interaction between the team, with at least one standing meeting every morning to discuss the days affairs and what should be deliverable or achieved by the end of it.

It is this element of Scrum that shall be utilized, albeit without the team aspect but rather, where testing feedback is involved, through the focus group setup to undertake critique of the final game and its interim deliverable beta releases for testing.

The below diagram is offered up to the reader to assist in visualization of the processes in Scrum and that also helps to see scrum as the fast paced and decisive development method it is.

G.Wright 2013

There is much more to Scrum and as can be seen it is a method in its own right, but fullest use of the methodology could be seen better described in a scenario involving a development team numbering greater than one.

To conclude, the use of both XP and Scrum items have been listed below, and which can be seen to eclectically pluck elements from each methodology to suit the needs of the development.

As such, the constraints that denote those aspects are as follows:

Single entity developer team, with only feedback encountered from the focus group at the high level.

Single entity design team for documentation of the system, with only feedback via the focus group at the high level.

This has denoted the aspects which must be taken from both paradigms and are as follows:

SCRUM

- Sprint planning: the team meets with the product owner to choose a set of work to deliver during a sprint
- Daily scrum: the team meets each day to share struggles and progress
- Sprint reviews: the team demonstrates to the product owner what it has completed during the sprint
- Sprint retrospectives: the team looks for ways to improve the product and the process.

XP

- Four principle activities – Coding, Testing, Listening, Designing
- XP Values- Communication, Simplicity, Feedback, Courage
- TDD

UML

- This will be the modelling language used to describe the system at any design stages, and is a common standard used in nearly all object oriented developments to allow high level users (the project focus group) to easily follow the system's architecture and processes.

2.5 Design Pattern Principles to Apply to the build

2.5.1 Factory Design Pattern used with prefabs

2.5.2 Singleton Design Pattern used with objects

2.5.3 Other Design Patterns used.

3 Project design

3.1 UML Documentation

3.1.1 Package Requirements

A skeletal infrastructure to hold a system in place that acts as a hub for interaction between a user and the system

3.1.2 Package diagram

The package diagram can be used to describe a high level view of the entire domain through separated packages that relate to the stripped down specifics of the system's requirements. Here one can view packages in similarity with entities at this high a level of abstraction.

Below has been offered the system as a package diagram:

3.1.2 Class Requirements

Component oriented designed class requirements

3.1.3 Class diagram

The main class diagram for the game development can be found in sub section 3.1.6 of this current section.

3.1.4 Class dependencies

The main class dependency diagram for the game development can be found in sub section 3.1.7 of this current section.

3.1.5 Class diagram Rich

Diagram of component classes and their linkages

3.1.6 Class Dependencies diagram

Diagram of dependant classes and any linkages

3.1.7 Object model dependencies

Diagram of object dependencies and any linkages

3.1.8 Use cases

Use cases describing system needs

3.1.9 Sequences

Game screen sequence diagram

Move player position sequence diagram

Player fire press sequence diagram

Collisions (player & projectile)

3.1.10 XP documentation

Using the XP practice, one can find code annotation placed throughout the games different classes that conform to the specification of XP to document the system in the code.

As such, please see the fully commented code for all classes at the rear of this assignment under Appendix A.

3.2 Game logic method explanantions

Using the XP practice, one can find code annotation placed throughout the games different classes that conform to the XP style of code documentation.

As such, please see the fully commented code that uses comments for method explanations at the rear of this assignment under Appendix A.

3.3 Release Planning

3.3.1 Release Plan & User stories

Please refer to the timeline located in Sub section 1.3.7 of Section 1 of which pertains to Gantt chart derived release planning, with respect to time constraints.

The following release plan collates User stories into the respective release packages and prior to an alpha, beta or final release the requirements of both the user and developer should be met. However, these should not be misinterpreted as the requirements specification, which was revealed within Section 1 during analysis & feasibility.

User Stories

Alpha release 1.0

As a user, I want to see how I control the game Conformance

- Keyboard
- Gamepad
- Touchpad
- Mouse
- Test unit until working

As a user, I want to see something react to my inputs Conformance

- (optional early insertion Player should have full movement
- Textual pop ups occur upon raycast toward a contextual object.
- Bones are collectable and itinerised on GUI
- Test unit until working

As a developer, I need to ensure player controls are sound Conformance

- Review controls until human similar actions are tweaked effectively, regarding player control mechanism.
- Test unit until working

As a developer, I need to ensure normals are correct for terrain collisions occurring from player, AI and interactive objects. Conformance

- Reduce terrain face intersection sharpness where possible
- Test unit until working

Alpha release 1.1

As a user, I want to see some of the game models and textures in the game Conformance

- Enemy sprite artwork completed
- Dumb AI (move from one side to the other toward player)
- Add collision detection for enemy and player collisions and player projectile hits.
- Test unit until working

As a developer, I need to establish interactions between AI and player Conformance

- Implement and refine target detection for all interactive AI elements in game.
- Test unit until working

Beta release 1.1

Final release

3.4 Planned Iterations testing

Alpha release 1.0

Main Iteration 1 Alpha release 1.1

Main Iteration 2 Beta release 1.0

Main Iteration 3 Final release 1.0

Main Iteration 4

Iteration delivery date 14/02/2013 21/02/2013 15/03/2013 11/05/2013

Responsibility Dev User Dev User Dev User Dev User

Test to be completed Test
structure Test
Controls &
Initial model testing Evaluate textures, Models,
Code Test
Controls &
Gameplay and textures

3.5 Final Test Table

3.5.2 Final Developer Approval Tests

3.5.3 Final User Approval Tests

3.5 UI Design

3.5.1 Start Screen UI

As can be seen in the image above, the UI controls are specified for the user to familiarize themselves with and by which, denotes how the user interfaces with the game at the start screen

3.5.2 In-Game UI

Below demonstrates the in-game UI, as seen by the user

3.5.3 Game over UI

3.6 Elements of Graphical Design

The introduction movie to the game was created with a composited mix of tools of which are briefly discussed and have been listed below.

Adobe Photoshop CS5 (All Artwork & art editing– hand drawn scans and digital)

Adobe Photoshop is viewed as a forefront, industry standard software for digital image and art editing on computer

Adobe After Effects CS6 (Potential for movie editing)

Adobe Soundbooth CS5 & FL Studio DAW (Audio mixing and editing)

3.6.1 Game Sequences (Unity Pro development)

The initial hand drawn story board for the introduction movie can be found at the rear of the project under Appendix E

3.6.2 Composite list of Scene Artwork and Models created

Player (FPS)

Torch

Kitty's Ghost

Kitty's Bones

Kitty's Grave

Houndtor Rock Scene

Kitty's Crossroads

Hound of the Baskervilles AI

CubeMaps

Particles

4 Fully Developed Game Code

4.1 Coded structure

4.1.1 Main Game Structure

The complete code for the class can be located at the back of this assignment under Appendix D entitled

4.2 Coded Base classes and Interfaces

4.2.1

The complete code for the class can be located at the back of this assignment under Appendix D entitled

4.2.2

The complete code for the class can be located at the back of this assignment under Appendix D entitled

4.2.3

The complete code for the class can be located at the back of this assignment under Appendix D entitled

4.2.4

The complete code for the class can be located at the back of this assignment under Appendix D entitled

4.2.4

The complete code for the class can be located at the back of this assignment under Appendix D entitled

4.3 Coded engine classes (if any???)

4.3.1

The complete code for the component class can be located at the back of this assignment under Appendix D entitled

4.3.2

The complete code for the ParticleEngine class can be located at the back of this assignment under Appendix D entitled

5 End User Documentation

5.1 Minimum Specifications

5.1.1 Windows PC

- Windows XP (x86) | Windows Vista (x86) –(x64) | Windows 7 (x86) –(x64) | Windows 8 (x86) –(x64).
- Acceptance of Administrator privileges may be required to install the standalone

Ram: 512 MB or more (Preferences – Min 1GB)

HDD: 1GB or more

Access to the internet for download from (as yet unspecified) website

5.1.1 Android (TBC)

All Android platforms V 4.0 Ice Cream Sandwich Upwards.???

5.2 User Manual

5.2.1 Install Windows

1. Unpack Zip file to a location of your choice.

2. Click once to install complete game
3. Game now installed on system
4. User controls can be found in the Start of the game menu

5.2.2 Install Android ? (TBC)

1. ?
2. ?
3. ?
4. ?
5. ?

6 Project Review

6.1 Critical Appraisal

The Analysis

The Design

The Build

6.2 Methodology Appraisal

7 Project Conclusions

7.2 Focus group Conclusion

7.2 What was achieved

7.3 Future developments

Appendix A

Appendix B

Appendix C

Focus Meeting Minutes 01/10/2012

Focus Meeting Minutes 10/12/2012

Focus Meeting Minutes 29/01/2013

Focus meeting minutes 15/02/2013

Appendix D

full code

Appendix E

Appendix F

Completed Gantt Chart

Completed Tracking Gantt Chart

Appendix G

Iteration test proof

Developer

G1

Gamestates: Intro, Start, InGame, & GameOver test screens

G2 & G3

Collision detection, projectile collision.

G4

Full game play, single pass screenshots

G5

Appendix H

Adobe Photoshop CS6

Adobe Soundbooth CS5 Used for some audio creations coming in the development

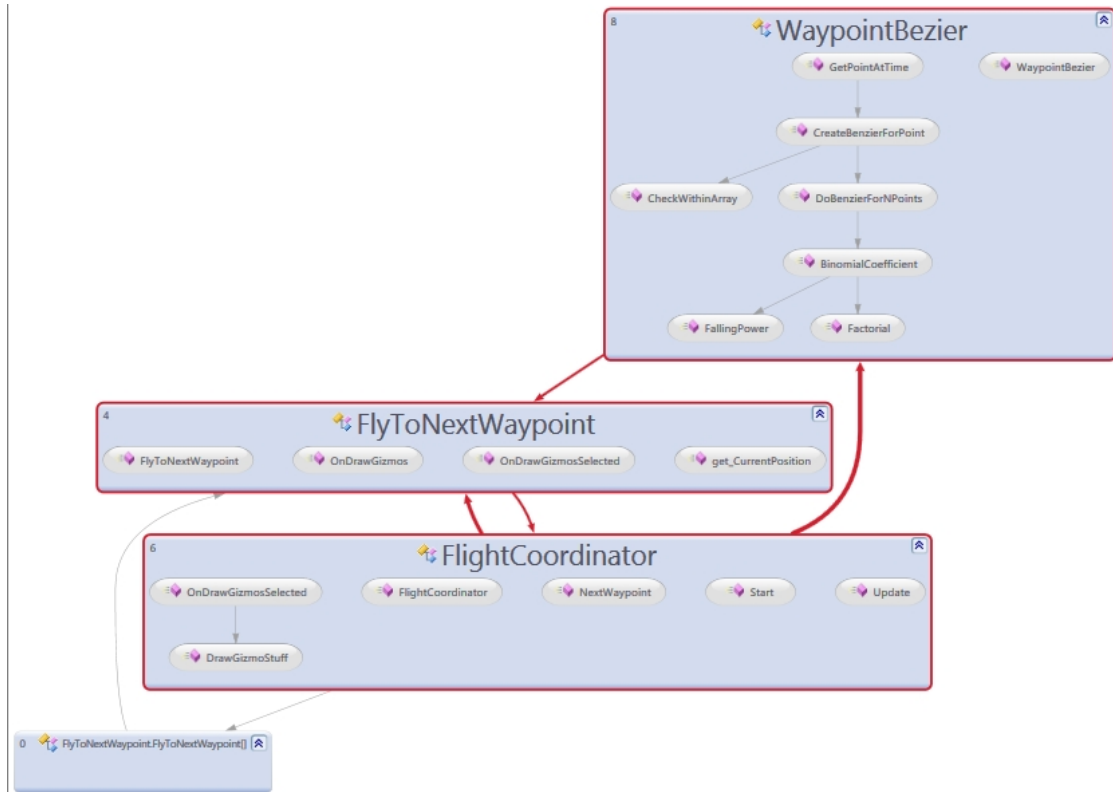
Imageline FL Studio 9 was also used to create audio adsssets like the dog bark noise amongst much more.

Appendix B

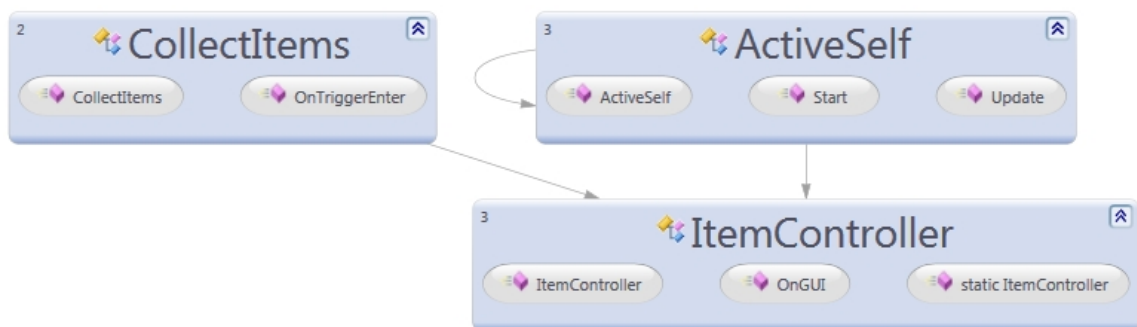
Dependencies

Class & Method

CameraSystem



Item controller



Object2Terrain

This semi-leveraged script(as some further conversion were brought more in line with C# by this developer) allowing a Terrain object to be prefabbed to a terrain object in Unity

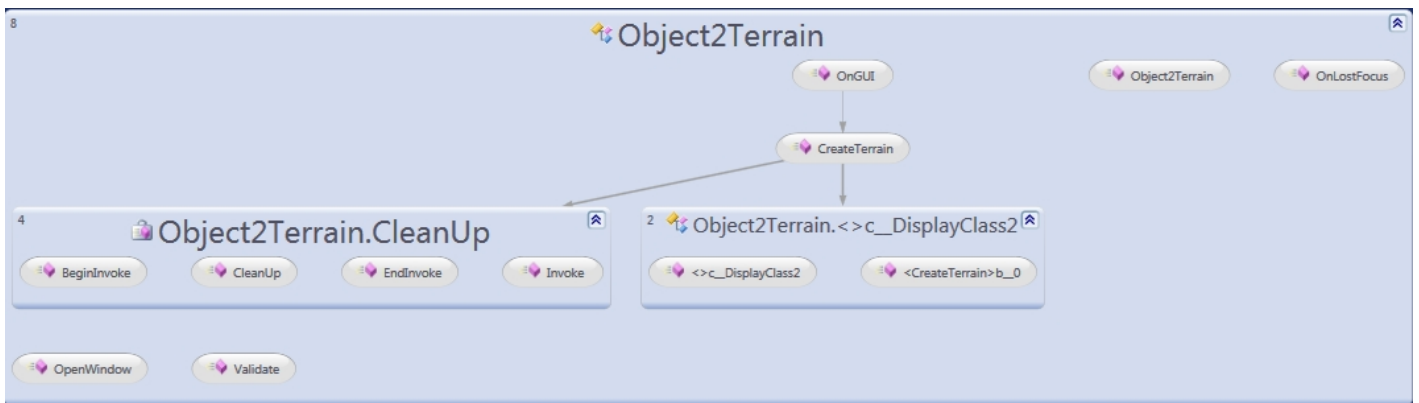
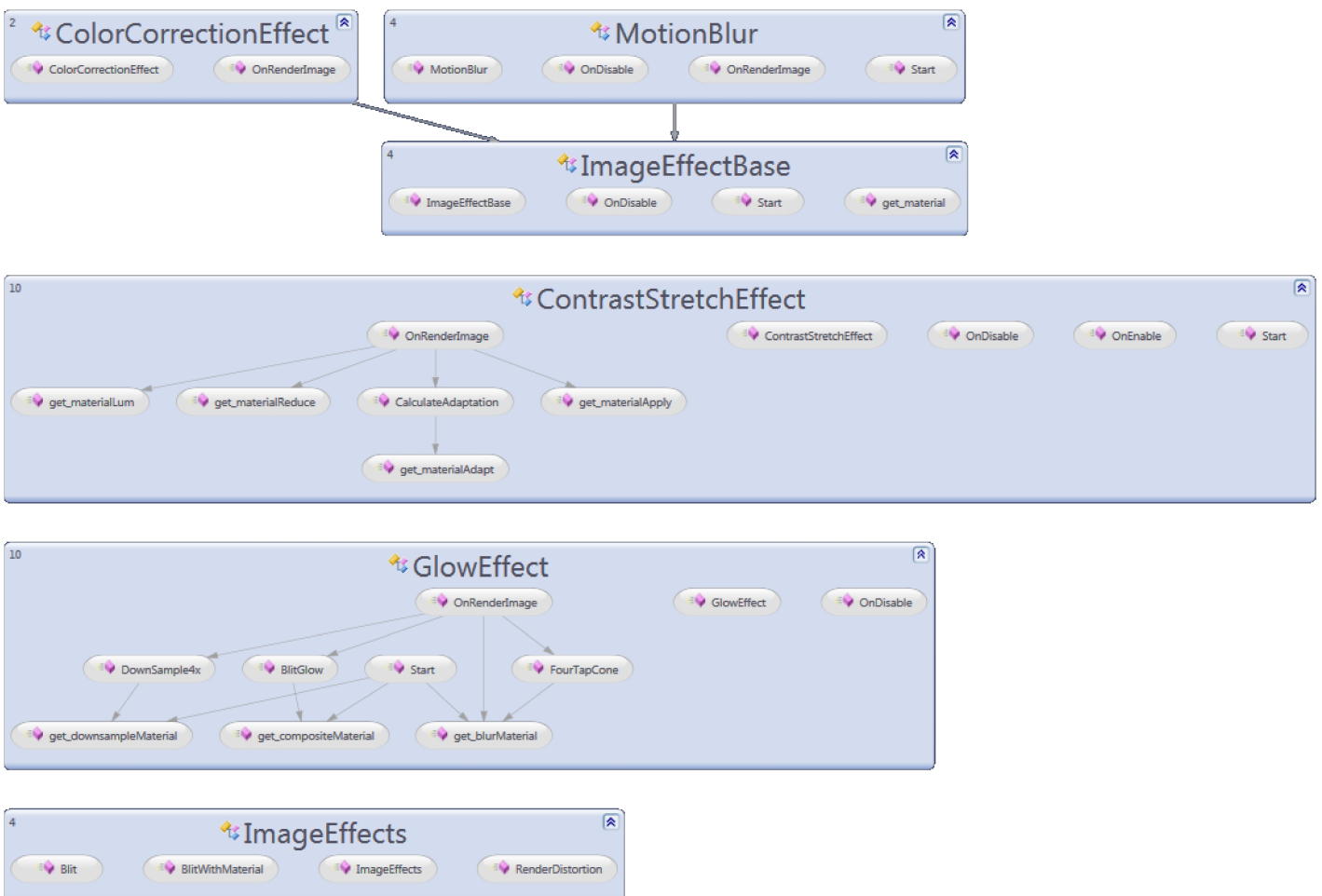
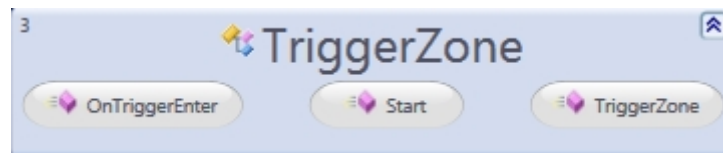


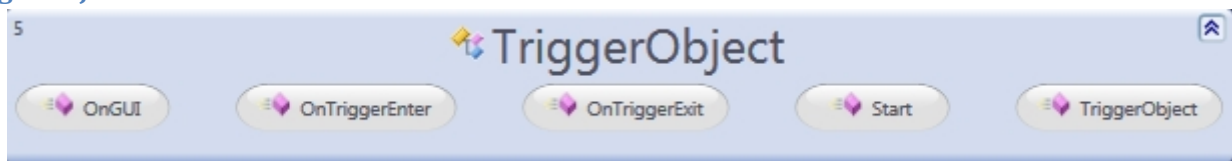
Image effects Complete



TriggerZone



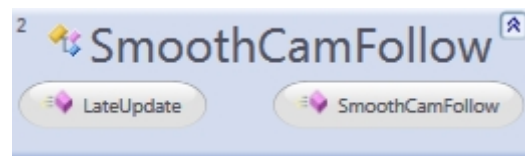
TriggerObject



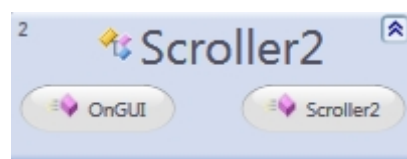
TriggerKittyAudio



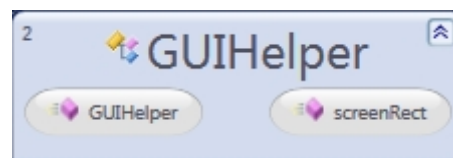
SmoothCamFollow



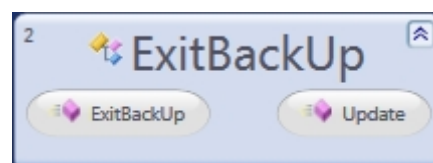
Scroller2



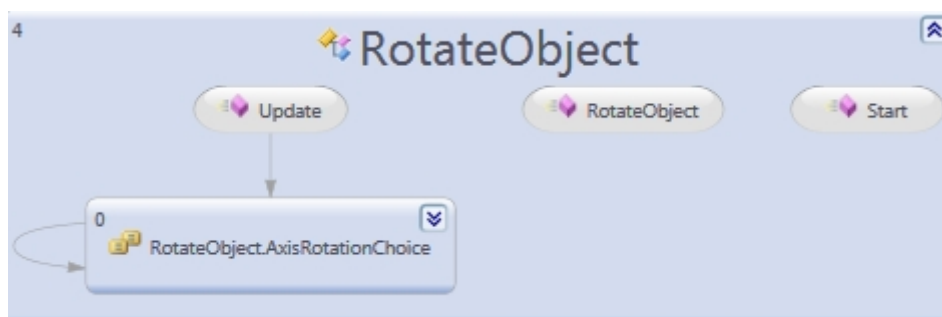
GUIHelper



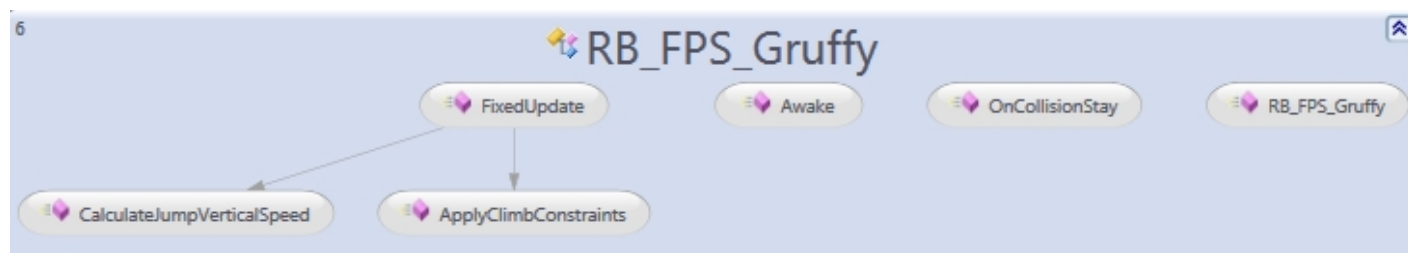
ExitBackUp



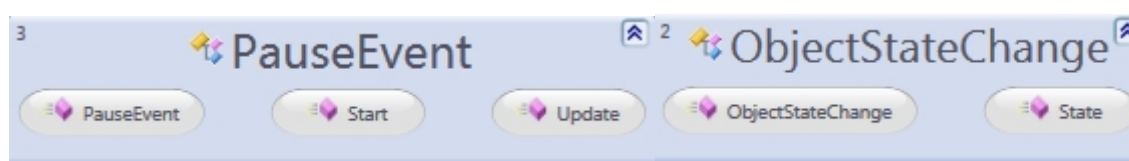
RotateObject



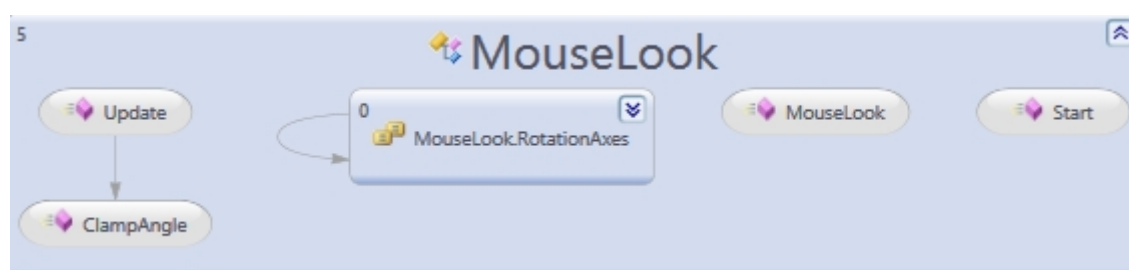
RB_FPS_Gruffy



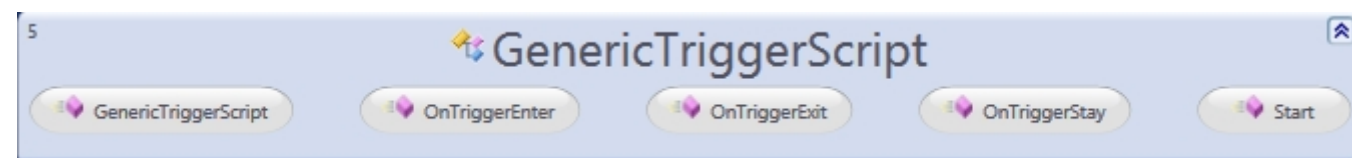
PauseEvent



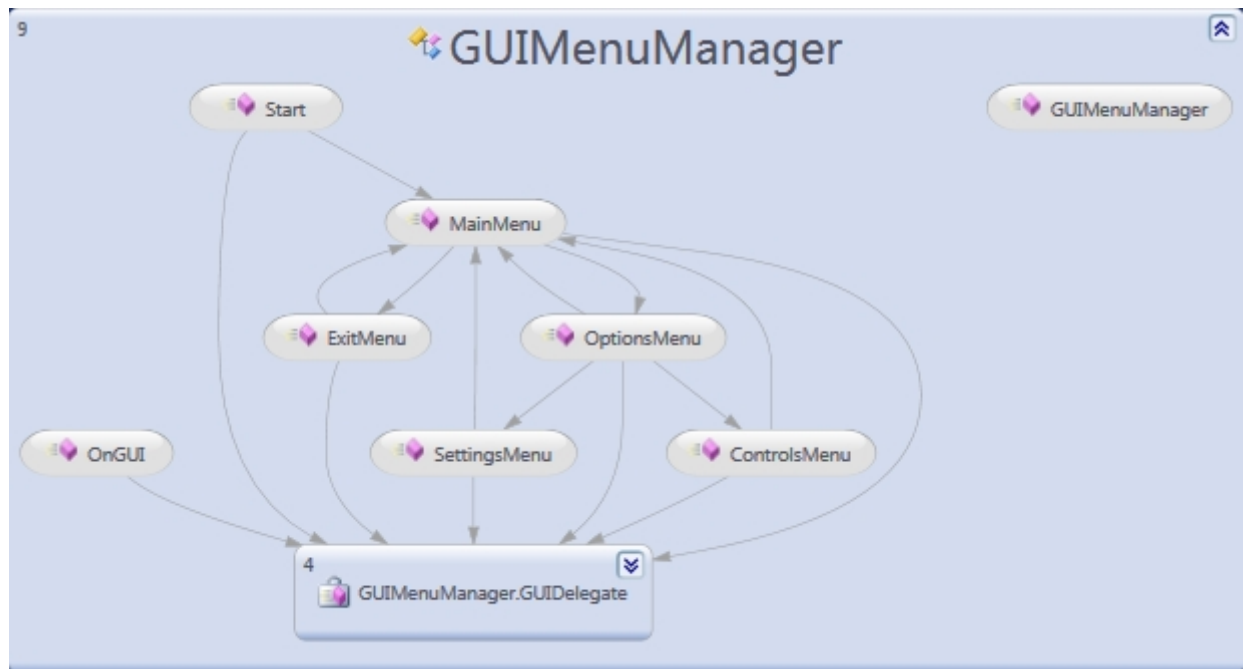
MouseLook



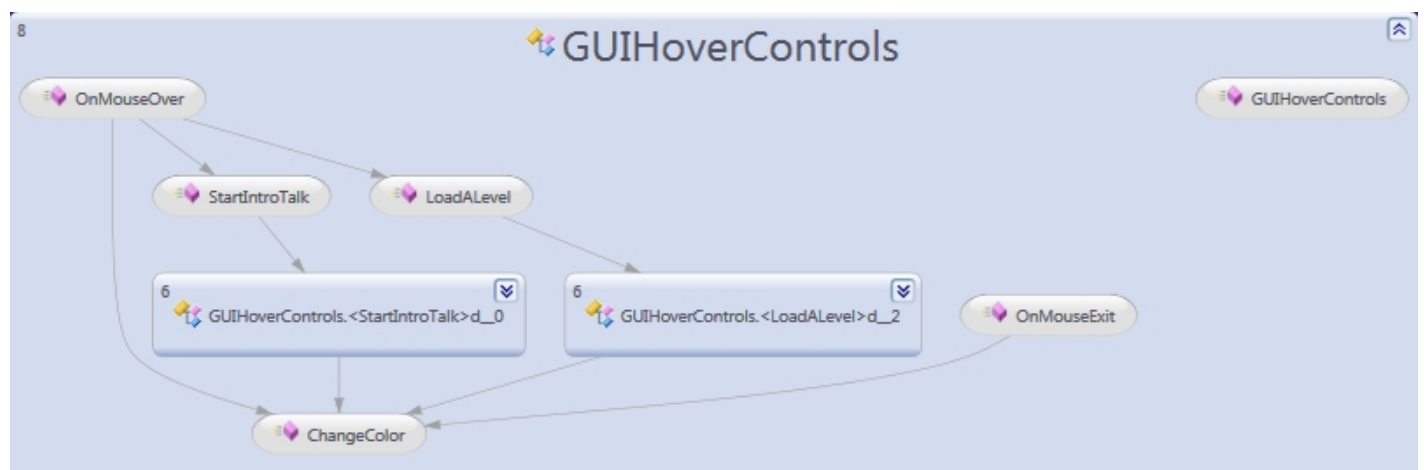
GenericTriggerScript



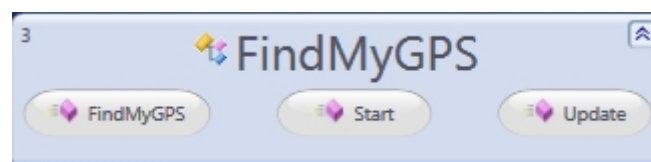
GUIMenuManager



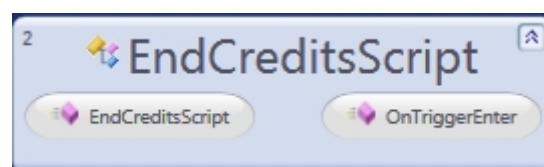
GUIHoverControls



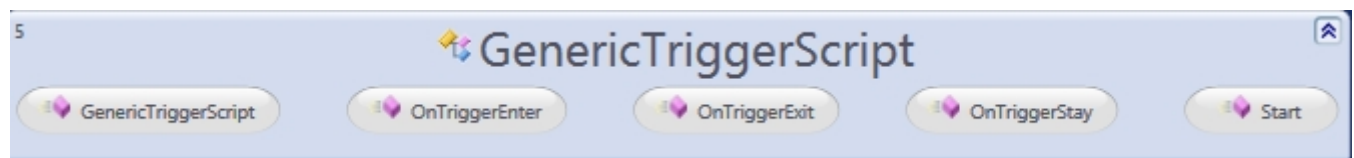
FindMyGPS



EndCreditsScript



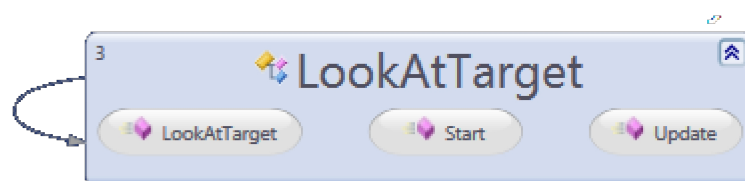
GenericTrigger



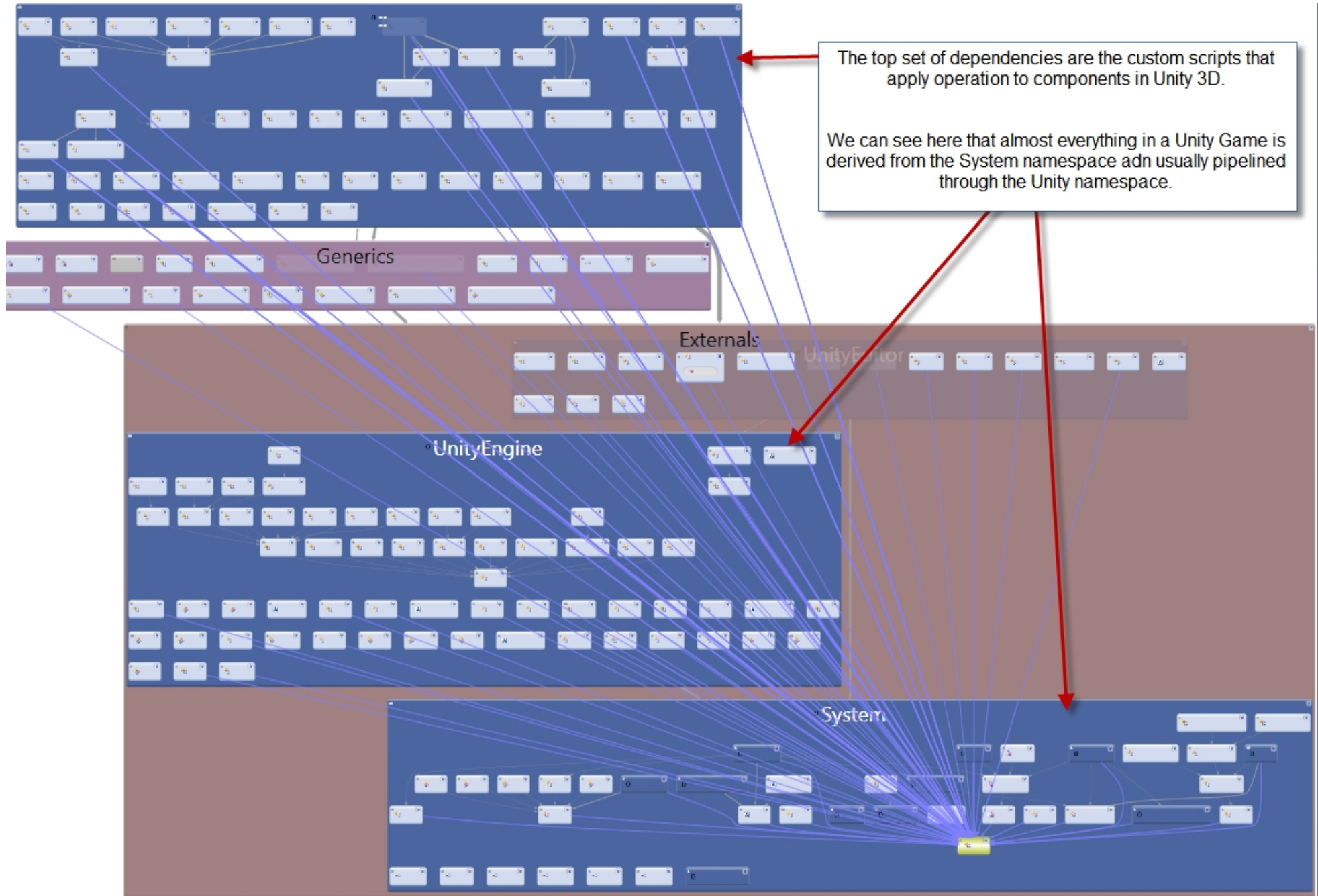
Teleporter



LookAtTarget



System Overview



Appendix C

Class Definitions and Diagrams

Below follow the class definitions and diagrams showing class attributes and operations

Name	Type	Modifier	Summary
Methods			
State	void	private	State the specified state.
<add method>			
Properties			
<add property>			
Fields			
go	GameObject[]	public	
<add field>			

ObjectStateCha...

Class

→ MonoBehaviour

Fields

go

Methods

State

Name	Type	Modifier	Summary
Methods			
OnTriggerEnter	void	private	Raises the trigger enter event.
OnTriggerExit	void	private	Raises the trigger exit event.
OnTriggerStay	void	private	Raises the trigger stay event.
Start	void	private	
<add method>			
Properties			
<add property>			
Fields			
alphaMin	bool	public	
gos	List<GameObject>	public	
target	GameObject	public	
<add field>			

GenericTriggerScript

Class

→ MonoBehaviour

Fields

alphaMin

gos

target

Methods

OnTriggerEnter

OnTriggerExit

OnTriggerStay

Start

Name	Type	Modifier	Summary
Methods			
OnTriggerEnter	void	private	Raises the trigger enter event.
Start	void	private	Start this instance.
<add method>			
Properties			
<add property>			
Fields			
go	GameObject	public	
msgObject	GameObject	public	

TriggerZone

Class

→ MonoBehaviour

Fields

go

msgObject

Methods

OnTriggerEnter

Start

Name	Type	Modifier	Summary
Methods			
DrawGizmoStuff	void	public	Draws the gizmo stuff.
NextWaypoint	Vector3	private	Nexths the waypoint.
OnDrawGizmosSelected	void	private	Raises the draw gizmos selected event.
Start	void	private	Start this instance.
Update	void	private	Update this instance.
<add method>			
Properties			
<add property>			
Fields			
bezier	WaypointBezier	private	current bezier cache
currentWaypoint	int	private	The current waypoint in flyNextWaypoints array
SecondsForFullLoop	float	public	The seconds for full loop.
targetPoint	Vector3	private	The target point.
targetRotation	Quaternion	private	The target rotation.
waypoints	FlyToNextWaypoir	private	

FlightCoordinator	Class
→ MonoBehaviour	
Fields	
bezier	
currentWaypoint	
SecondsForFull...	
targetPoint	
targetRotation	
waypoints	
Methods	
DrawGizmoStuff	
NextWaypoint	
OnDrawGizmos...	
Start	
Update	

Name	Type	Modifier	Summary
Methods			
OnDrawGizmos	void	private	Raises the draw gizmos event.
OnDrawGizmosSelected	void	private	Raises the draw gizmos selected event.
<add method>			
Properties			
CurrentPosition	Vector3	public	Gets the current position.
<add property>			
Fields			
<add field>			

FlyToNextWayp...	Class
→ MonoBehaviour	
Properties	
CurrentPosition	
Methods	
OnDrawGizmos	
OnDrawGizmos...	

Name	Type	Modifier	Summary
Methods			
BinomialCoefficient	float	private	Finds the Binomials coefficient.
CheckWithinArray	int	private	Checks the within array.
CreateBezierForPoint	Vector3	private	Creates the bezier for point.
DoBezierForNPoints	Vector3	private	finds the bezier for N points.
Factorial	int	private	Finds Factorial of specified n.
FallingPower	int	private	Falling power.
GetPointAtTime	Vector3	public	Gets the point at time.
WaypointBezier		public	Initializes a new instance of the <see cref="WaypointBezier"/> class.
<add method>			
Properties			
<add property>			
Fields			
vectorArray	Vector3[]	private	

WaypointBezier	Class
→ MonoBehaviour	
Fields	
vectorArray	
Methods	
BinomialCoefficient	
CheckWithinArray	
CreateBezierForPoint	
DoBezierForNPoints	
Factorial	
FallingPower	
GetPointAtTime	
WaypointBezier	

Name	Type	Modifier	Summary
Methods			
LateUpdate <add method>	void	private	Late update.
Properties			
<add property>			
Fields			
distance	float	public	The distance in the x-z plane to the target
height	float	public	the height we want the camera to be above the target
heightDamping	float	public	The height damping.
rotationDamping	float	public	The rotation damping.
target	Transform	public	The target we are following
targetPoint	Vector3	private	The target point.
targetRotation	Quaternion	private	The target rotation.

SmoothCamFoll...

Class

→ MonoBehaviour

Fields

distance
 height
 heightDamping
 rotationDamping
 target
 targetPoint
 targetRotation

Methods

LateUpdate

Name	Type	Modifier	Summary
Methods			
Start <add method>	void	private	Start this instance.
Update <add method>	void	private	Update this instance.
Properties			
<add property>			
Fields			
inst	LookAtTarget	private	The instanc of itself
target	GameObject	public	The target is an empty gameobject behind the player
targetPoint	Vector3	private	The target point-to look at
targetRotation	Quaternion	private	The target rotation to rotate toward
<add field>			

LookAtTarget

Class

→ MonoBehaviour

Fields

inst
 target
 targetPoint
 targetRotation

Methods

Start
 Update

Name	Type	Modifier	Summary
Methods			
Start	void	private	Start this instance.
Update	void	private	Update this instance.
<add method>			
Properties			
<add property>			
Fields			
inst	LookAtTarget	private	The instanc of itself
target	GameObject	public	The target is an empty gameobject behind the player.
targetPoint	Vector3	private	The target point-to look at
targetRotation	Quaternion	private	The target rotation to rotate toward
<add field>			

Name	Type	Modifier	Summary
Methods			
OnGUI	void	private	Raises the GU event.
OnTriggerEnter	void	private	Raises the trigger enter event.
OnTriggerExit	void	private	Raises the trigger exit event.
Start	void	private	Start this instance.
<add method>			
Properties			
<add property>			
Fields			
changeToMaterial	Material	public	The change to material.
content	List<string>	public	
coords	List<Vector2>	public	The coords.
distance	int	public	The distance.
go	GameObject	private	The go.
objectToChange	GameObject	public	The object to change.
originalMaterial	Material	public	The original material.
style	GUIStyle	public	The style.
triggerFieldIs	bool	public	The trigger field is.

LookAtTarget

Class
→ MonoBehaviour

Fields

- inst
- target
- targetPoint
- targetRotation

Methods

- Start
- Update

TriggerObject

Class
→ MonoBehaviour

Fields

- changeToMater...
- content
- coords
- distance
- go
- objectToChange
- originalMaterial
- style
- triggerFieldIs

Methods

- OnGUI
- OnTriggerEnter
- OnTriggerExit
- Start

Name	Type	Modifier	Summary
Methods			
Start	void	private	Start this instance.
<add method>			
Properties			
<add property>			
Fields			
Bump0	Texture2D	public	The bump0.1.2.3
Bump1	Texture2D	public	
Bump2	Texture2D	public	
Bump3	Texture2D	public	
ColTex0	Color	public	The col tex0.1.2.3
ColTex1	Color	public	
ColTex2	Color	public	
ColTex3	Color	public	
CustomColorMap	Texture2D	public	The custom color map.
SplattingDistance	float	public	The splatting distance.
TerrainNormalMap	Texture2D	public	The terrain normal map.

CustomTerrainS...
Class
→ MonoBehaviour

Fields

- Bump0
- Bump1
- Bump2
- Bump3
- ColTex0
- ColTex1
- ColTex2
- ColTex3
- CustomColorM ...
- SplattingDistan ...
- TerrainNormal...

Methods

- Start

Name	Type	Modifier	Summary
Methods			
Start	void	private	Start this instance.
Update	void	private	Update this instance.
<add method>			
Properties			
<add property>			
Fields			
axisRotationChoice	AxisRotationChoic	public	
axisRotChoice	AxisRotationChoic	private	
chosenSpeed	float	public	
variants	float[]	public	
<add field>			

RotateObject
Class
→ MonoBehaviour

Fields

- axisRotationChoice
- axisRotChoice
- chosenSpeed
- variants

Methods

- Start
- Update

Nested Types

AxisRotationC...
Enum

- AntiClockWiseY
- ClockWiseY
- AntiClockWiseX
- ClockWiseX
- AntiClockWiseZ
- ClockWiseZ
- AntiClockWiseMix
- ClockWiseMix

Enumeration choices to signify rotation control

Name	Type	Modifier	Summary
Methods			
▶ OnGUI	void	private	Raises the GUI event.
<add method>			
Properties			
<add property>			
Fields			
offset	float	public	
scrollLines	string[]	public	
speed	float	public	
style	GUIStyle	public	
<add field>			

Scroller2

Class

→ MonoBehaviour

Fields

- offset
- scrollLines
- speed
- style

Methods

- OnGUI

Name	Type	Modifier	Summary
Methods			
▶ ApplyClimbConstraints	void	private	Applies the climb constraints.
▶ Awake	void	private	Awake this instance.
▶ CalculateJumpVerticalSpeed	float	private	Calculates the jump vertical speed.
▶ FixedUpdate	void	private	Fixed the update.
▶ OnCollisionStay	void	private	Raises the collision stay event.
<add method>			
Properties			
<add property>			
Fields			
audioClip	AudioClip	public	
canJump	bool	public	
canRun	bool	public	
canRunSidestep	bool	public	
gravity	float	public	
grounded	bool	private	
inAirControl	float	public	
isTouchEnabled	bool	public	
jumpHeight	float	public	
maxVelocityChange	float	public	
runBackwardSpeed	float	public	
runSidestepSpeed	float	public	
runSpeed	float	public	
sidestepSpeed	float	public	
slideDir	Vector3	private	
slideMagnitudeConstraint	float	public	
slidingThreshold	float	public	
targetVelocity	Vector3	private	
velocity	Vector3	private	
velocityChange	Vector3	private	
walkBackwardSpeed	float	public	
walkSpeed	float	public	
<add field>			

RB_FPS_Gruffy

Class

→ MonoBehaviour

Fields

- audioClip
- canJump
- canRun
- canRunSidestep
- gravity
- grounded
- inAirControl
- isTouchEnabled
- jumpHeight
- maxVelocityCh...
- runBackwardSp...
- runSidestepSpe...
- runSpeed
- sidestepSpeed
- slideDir
- slideMagnitude...
- slidingThreshold
- targetVelocity
- velocity
- velocityChange
- walkBackwardS...
- walkSpeed

Methods

- ApplyClimbCon...
- Awake
- CalculateJump...
- FixedUpdate
- OnCollisionStay

Name	Type	Modifier	Summary
Methods			
▶ ClampAngle	float	public	
▶ Start	void	private	
▶ Update	void	private	
<add method>			
Properties			
<add property>			
Fields			
axes	RotationAxes	public	
maximumX	float	public	
maximumY	float	public	
minimumX	float	public	
minimumY	float	public	
originalRotation	Quaternion	private	
rotationX	float	private	
rotationY	float	private	
sensitivityX	float	public	
sensitivityY	float	public	
<add field>			

MouseLook

Class

→ MonoBehaviour

Fields

- axes
- maximumX
- maximumY
- minimumX
- minimumY
- originalRotation
- rotationX
- rotationY
- sensitivityX
- sensitivityY

Methods

- ClampAngle
- Start
- Update

Nested Types

RotationA...

Enum

- MouseXAndY
- MouseX
- MouseY

Enum to signify
mouse axis
control

Name	Type	Modifier	Summary
Methods			
Start	void	private	Start this instance.
Update	void	private	Update this instance.
<add method>			
Properties			
<add property>			
Fields			
go	GameObject	public	The go.
lastPos	Vector3	private	The last position.
mylatitude	float	public	The mylatitude.
mylongitude	float	public	The mylongitude.
myradius	float	public	The myradius.
objWarning	string	private	The object warning.
playerPos	Vector3	public	The player position.
xPos	float	private	The x position.
yPos	float	private	The x position.
zPos	float	private	The x position.

FindMyGPS
 Class
 ↳ MonoBehaviour

Fields

- go
- lastPos
- mylatitude
- mylongitude
- myradius
- objWarning
- playerPos
- xPos
- yPos
- zPos

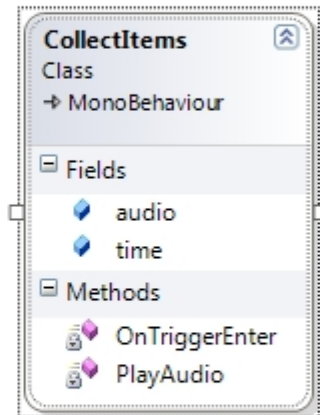
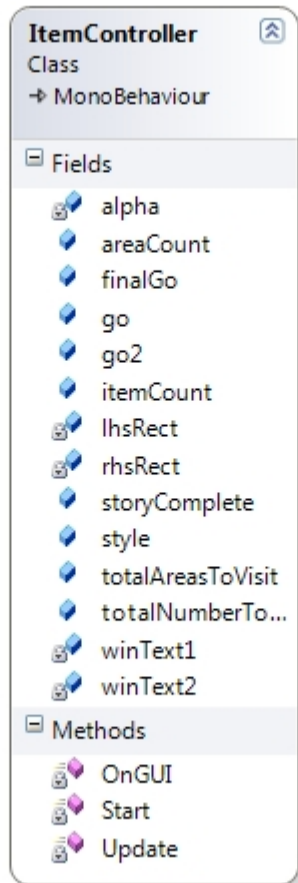
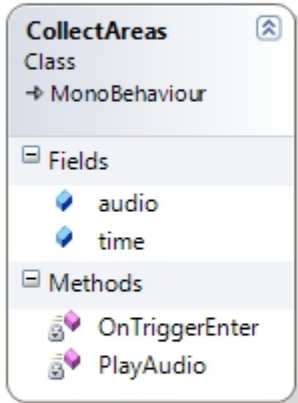
Methods

- Start
- Update

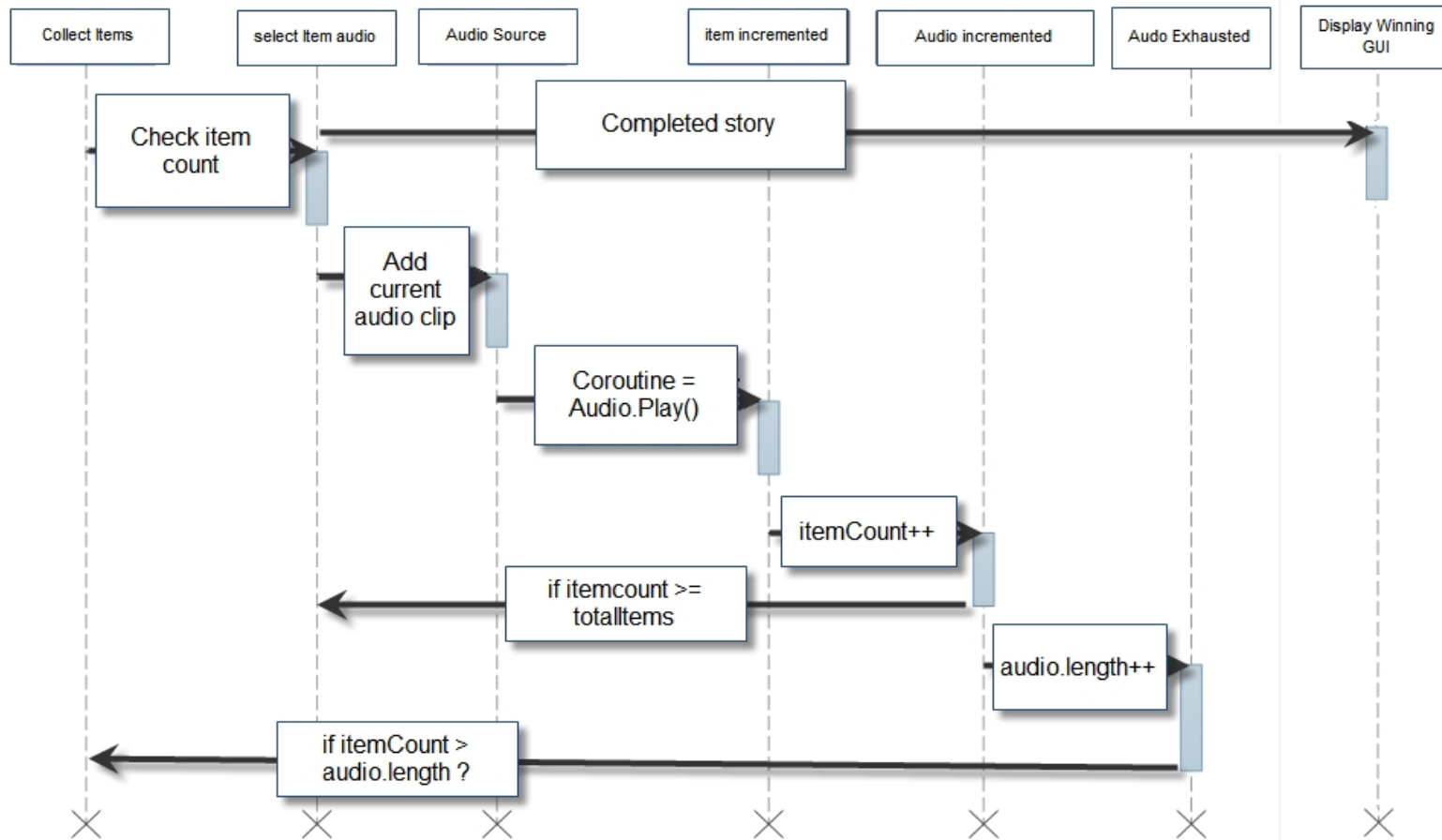
Not yet fully implemented but in as a future development script

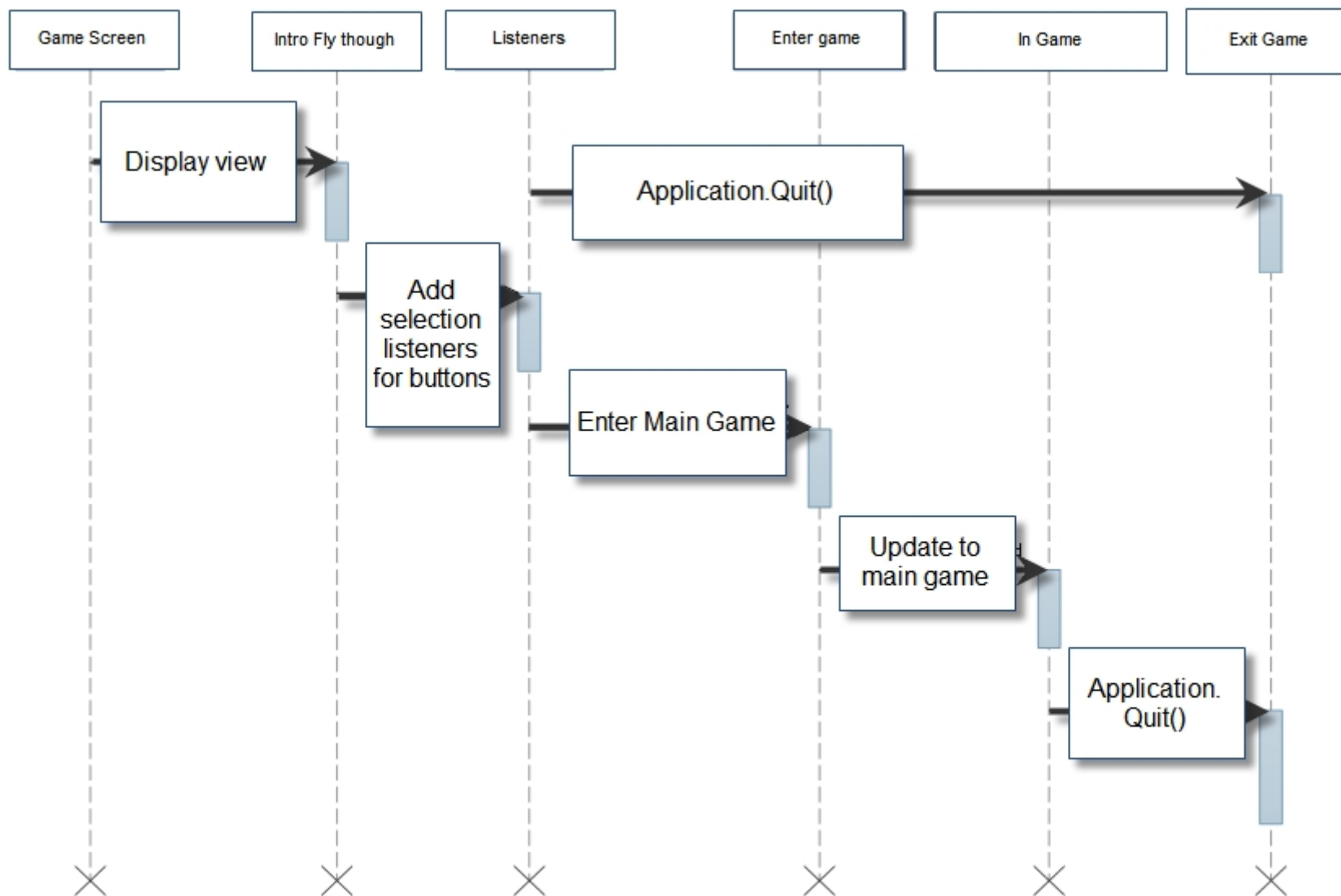
These, component connected, class diagrams deal with player controls and the future GPS handler (though this is working it is not fully implemented and is a future development for later release – the concept was proved in game by discovery of geo caches in the area, the find GPS script will be later used to employ an internal game compass to make finding them more interesting)

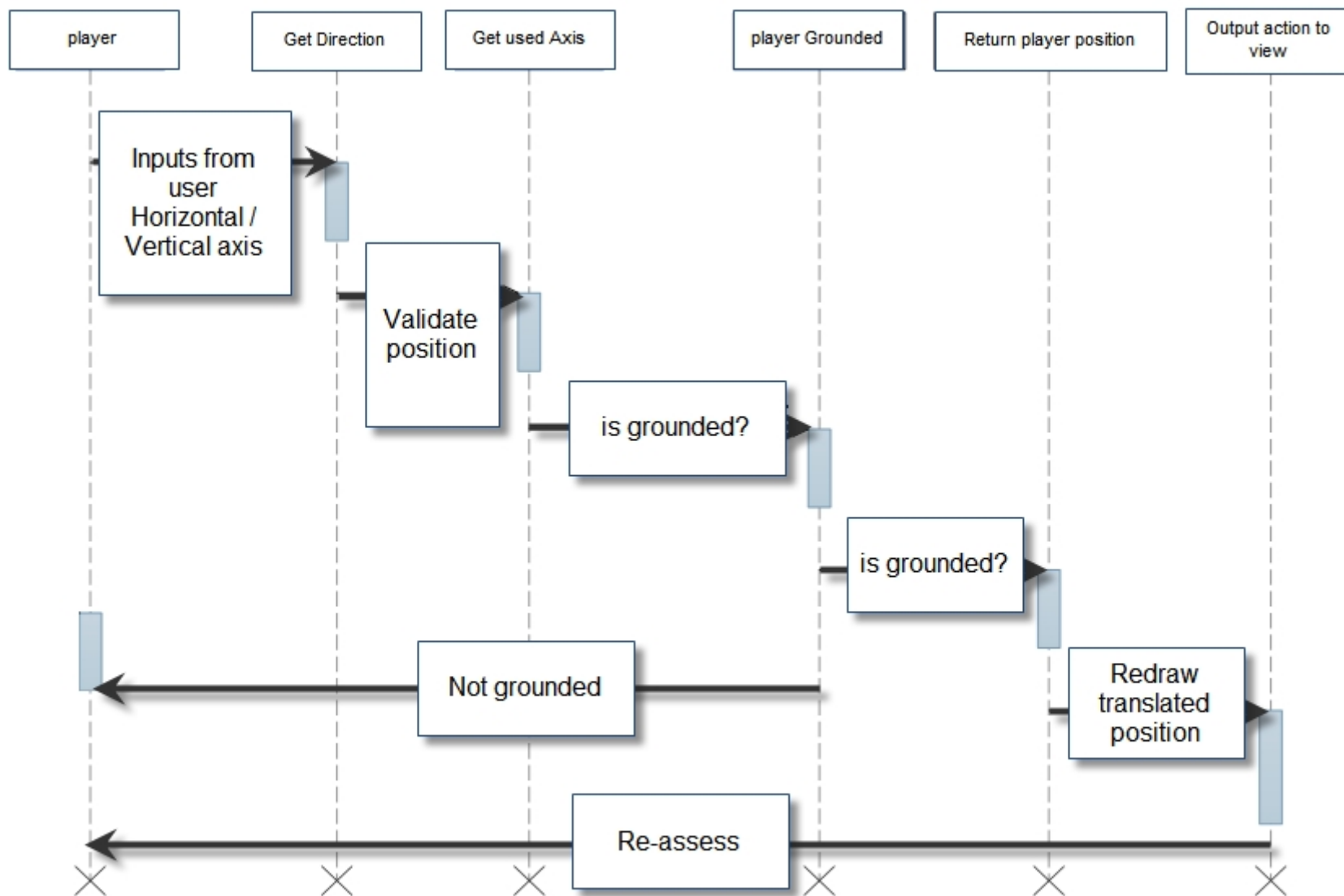
<p>Still to add</p>	<div data-bbox="804 125 1082 499"><p>PauseEvent Class → MonoBehaviour</p><p>Fields</p><ul style="list-style-type: none">pausedtargetObj<p>Methods</p><ul style="list-style-type: none">StartUpdate</div> <div data-bbox="1206 125 1484 947"><p>GUIMenuManag... Class → MonoBehaviour</p><p>Fields</p><ul style="list-style-type: none">aClipcurDelegatefogDistancegammanextstyleOfChoiceuniformSkinxPosyPos<p>Methods</p><ul style="list-style-type: none">ControlsMenuExitMenuMainMenuOnGUIOptionsMenuSettingsMenuStart<p>Nested Types</p></div>
<p>Still to add</p>	<div data-bbox="804 992 1098 1225"><p>EndCreditsScript Class → MonoBehaviour</p><p>Methods</p><ul style="list-style-type: none">OnTriggerEnter</div> <div data-bbox="1139 992 1439 1225"><p>ExitBackUp Class → MonoBehaviour</p><p>Methods</p><ul style="list-style-type: none">Update</div>
<p>Still to add</p>	<div data-bbox="804 1252 1088 2042"><p>GUIHoverControls Class → MonoBehaviour</p><p>Fields</p><ul style="list-style-type: none">beepfadedOutMatisClickedisQuitButtonlevelToLoadmat1mat2mat3narrativetxtMeshwait<p>Methods</p><ul style="list-style-type: none">ChangeColorLoadALevelOnMouseExitOnMouseOverStartIntroTalk</div>



Sequence Diagrams







Appendix D

XP Documented Game Code

SmoothCamFollow.cs

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Smooth cam follow.
/// Converted by Gruffy (Gareth Wright)2013 from the Unity Scripts "SmoothFollow.js"
script
/// Rewritten in Csharp to conform to the project's CSharp only program code rule.
/// This camera smoothes out rotation around the y-axis and height.
/// Horizontal Distance to the target is always fixed.
/// There are many different ways to smooth the rotation but doing it this way gives
you a lot of control over how the camera behaves.
/// For every of those smoothed values we calculate the wanted value and the current
value.
/// Then we smooth it using the Lerp function.
/// Then we apply the smoothed values to the transform's position.
/// </summary>
public class SmoothCamFollow : MonoBehaviour
{
    /// <summary>
    /// The target we are following
    /// </summary>
    public Transform target;
    /// <summary>
    /// The distance in the x-z plane to the target
    /// </summary>
    public float distance = 10.0f;
    /// <summary>
    /// the height we want the camera to be above the target
    /// </summary>
    public float height = 5.0f;
    /// <summary>
    /// The height damping.
    /// </summary>
    public float heightDamping = 2.0f;
    /// <summary>
    /// The rotation damping.
    /// </summary>
    public float rotationDamping = 3.0f;
    /// <summary>
    /// The target point.
    /// </summary>
    private Vector3 targetPoint;
    /// <summary>
    /// The target rotation.
    /// </summary>
    private Quaternion targetRotation;
    /// <summary>
    /// Late update.
    /// find the target and set the rotation to turn toward
    /// do same for world height of transform
    /// aim this transforms look direction at the target transform
    /// dampen correctional positoning and rotation
    /// </summary>
    void LateUpdate ()
    {
        if (!target)
            return;
    }
}
```

```

float wantedRotationAngle = target.eulerAngles.y;
float wantedHeight = target.position.y + height;

float currentRotationAngle = transform.eulerAngles.y;
float currentHeight = transform.position.y;

currentRotationAngle = Mathf.LerpAngle (currentRotationAngle, wantedRotationAngle,
rotationDamping * Time.deltaTime); /// Damp the rotation around the y-axis

currentHeight = Mathf.Lerp (currentHeight, wantedHeight, heightDamping *
Time.deltaTime); /// Damp the height

var currentRotation = Quaternion.Euler (0, currentRotationAngle, 0); /// Convert
the angle into a rotation

transform.position = target.position; /// Set the position of the camera on the x-z
plane to:
transform.position -= currentRotation * Vector3.forward * distance; /// distance
meters behind the target

transform.position = new Vector3(transform.position.x, currentHeight,
transform.position.z); /// Set the height of the camera

transform.LookAt (target); /// Always look at the target - bad use of Unity in
built heavy method
    }
}

```

LookAtTarget.cs

```

using UnityEngine;
using System.Collections;
/// <summary>
/// Look at target.
/// checks the gameobject's current transform position
/// and rotates toward face the target (-targetPoint)
/// </summary>
public class LookAtTarget : MonoBehaviour
{
    /// <summary>
    /// The target is an empty gameobject behind the player transform as onintersection
    with a collider the LookAt function has issues
    /// </summary>
    public GameObject target;
    /// <summary>
    /// The instanc of itself
    /// </summary>
    private LookAtTarget inst;
    /// <summary>
    /// The target point-to look at
    /// </summary>
    private Vector3 targetPoint;
    /// <summary>
    /// The target rotation to rotate toward
    /// </summary>
}

```

```

    /// <summary>
    /// The target rotation - the new quaternion to rotate toward.
    /// </summary>
    private Quaternion targetRotation;
    /// <summary>
    /// Start this instance.
    /// set inst to this gameobject instance
    /// </summary>
    void Start ()
    {
        inst = this;
    }

    /// <summary>
    /// Update this instance.
    /// set new target vector for rotation
    /// rotate to target
    /// apply to instance
    /// </summary>
    void Update ()
    {
        targetPoint = new Vector3(target.transform.position.x, transform.position.y,
target.transform.position.z) - transform.position;
        targetRotation = Quaternion.LookRotation (-targetPoint, Vector3.up);
        inst.gameObject.transform.rotation = Quaternion.Slerp(transform.rotation,
targetRotation, 0.1f);
    }

```

FlightCoordinator.cs

```

/* Adapted from a Sam Cox JavaScript - 2009 - FrictionPointStudios.com
 * C# conversion, documentation and optimization by G.Wright 2013
 * - Final adapted Code utilises LookRotation instead of the heavy LookAt() Unity
Engine method
 */

using UnityEngine;
using System.Collections;
/// <summary>
/// Flight coordinator.
/// </summary>
public class FlightCoordinator: MonoBehaviour
{
    //array instance of waypoint
    private FlyToNextWaypoint[] waypoints;
    /// <summary>
    /// current bezier cache
    /// </summary>
    private WaypointBezier bezier;
    /// <summary>
    /// The current waypoint in flyNextWaypoints array
    /// </summary>
    int currentWaypoint = 0;
    /// <summary>
    /// The seconds for full loop.
    /// Holds a time value to complete a full loop by
    /// </summary>
    public float SecondsForFullLoop = 60f;
    /// <summary>
    /// The target point.
    /// </summary>
    private Vector3 targetPoint;
    /// <summary>
    /// The target rotation.
    /// vector to cache to the targets axis of rotation

```

```

    /// </summary>
    private Quaternion targetRotation;
    /// <summary>
    /// Start this instance.
    /// component array made equal to children transforms(wayPoints) in the parent
gameobject
    /// the script is attached to...
    /// Waypoints[] is instanced at each call with the current length of waypoints left
    /// and it and its index are copied
    /// Using optimized LookRotation instead of heavy LookAt() method
    /// </summary>
    void Start ()
    {
        Component[] rawArray =
transform.parent.GetComponentsInChildren(typeof(FlyToNextWaypoint));
        if (rawArray.Length == 0)
        {
            Debug.LogError("You have no 'FlyToNextWaypoint' objects defined. Create an
empty object at the same level as this and add the FlyToNextWaypoint script to it");
        }
        waypoints = new FlyToNextWaypoint[rawArray.Length];
        rawArray.CopyTo(waypoints, 0);
        bezier = new WaypointBezier(waypoints);
        transform.position = waypoints[0].CurrentPosition;
        Vector3 newPosition = bezier.GetPointAtTime(0.01f);

        targetPoint = new Vector3(newPosition.x, newPosition.y, newPosition.z) -
transform.position;
        targetRotation = Quaternion.LookRotation (targetPoint, Vector3.up);
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation,
0.1f);
    }
    /// <summary>
    /// Update this instance.
    /// sets a timestamp per cycle
    /// assigns the vector positon to the current bezier way point/position
    /// rotates to look at transform
    /// assigns current transform position to the next bezier point/position
    /// Using optimized LookRotation instead of heavy LookAt() method
    /// </summary>
    void Update ()
    {
        float currentTime = Time.time % SecondsForFullLoop;
        Vector3 newPosition = bezier.GetPointAtTime(currentTime / SecondsForFullLoop);

        targetPoint = new Vector3(newPosition.x, newPosition.y, newPosition.z) -
transform.position;
        targetRotation = Quaternion.LookRotation (targetPoint, Vector3.up);
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation,
0.1f);

        transform.position = newPosition;
    }
    /// <summary>
    /// Nexts the waypoint.
    /// </summary>
    /// <returns>
    /// The waypoint.
    /// </returns>
    private Vector3 NextWaypoint()
    {
        if (currentWaypoint + 1 > waypoints.Length - 1)
        {

```

```

        return waypoints[0].CurrentPosition;
    }
    else
    {
        return waypoints[currentWaypoint + 1].CurrentPosition;
    }
}

/// <summary>
/// Raises the draw gizmos selected event.
/// </summary>
void OnDrawGizmosSelected()
{
    DrawGizmoStuff();
}

/// <summary>
/// Draws the gizmo stuff.
/// This is in a separate method because the waypoints themselves will also call it
when selected
/// this is an editor extension so the camera route can be placed visually
/// </summary>
public void DrawGizmoStuff()
{
    Component[] rawArrayGiz;
    FlyToNextWaypoint[] waypointsGiz;

    rawArrayGiz =
transform.parent.GetComponentsInChildren(typeof(FlyToNextWaypoint));
    waypointsGiz = new FlyToNextWaypoint[rawArrayGiz.Length];
    rawArrayGiz.CopyTo(waypointsGiz, 0);

    //debug error log
    WaypointBezier waypointBezierGiz = new WaypointBezier(waypointsGiz);

    for (float t = 0; t < 1f; t += 0.001f)
    {
        Gizmos.DrawIcon(waypointBezierGiz.GetPointAtTime(t), "WaypointHeading.tif");
    }
}
}

```

FlyToNextWaypoint.cs

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Fly to next waypoint.
/// C# , documentation and reformatting to suit my needs better by G.Wright 2013 -
Final Project Code
/// </summary>
public class FlyToNextWaypoint : MonoBehaviour
{
    /// <summary>
    /// Gets the current position.
    /// </summary>
    /// <value>
    /// The current position = this transforms position
    /// </value>
    public Vector3 CurrentPosition
    {
        get
        {
            return transform.position;
        }
    }
    /// <summary>
    /// Raises the draw gizmos event.
    /// </summary>
    void OnDrawGizmos()
    {
        Gizmos.DrawIcon(transform.position, "Waypoint.tif");
    }
    /// <summary>
    /// Raises the draw gizmos selected event.
    /// keeps the visual refernce when selcting child objects of gameobject with
    flightcoordinator attached script
    /// </summary>
    void OnDrawGizmosSelected()
    {
        (transform.parent.GetComponentInChildren(typeof(FlightCoordinator)) as
        FlightCoordinator).DrawGizmoStuff();
    }
}
```

WaypointBezier.cs

```
/* Original JavaScript by Sam Cox - 2009 - FrictionPointStudios.com
 * C# conversion, documentation and reformatting to suit my needs better by G.Wright
 2013 - Final Project Code
 */
```

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Waypoint bezier.
/// </summary>
public class WaypointBezier : MonoBehaviour
{
    Vector3[] vectorArray;
    /// <summary>
    /// Initializes a new instance of the <see cref="WaypointBezier"/> class.
    /// </summary>
    /// <param name='waypointsArray'>
    /// Waypoints array.
    /// </param>
    public WaypointBezier(FlyToNextWaypoint[] waypointsArray)
    {
        vectorArray = new Vector3[waypointsArray.Length];

        for (int i = 0; i < waypointsArray.Length; i++)
        {
            vectorArray[i] = waypointsArray[i].CurrentPosition;
        }
    }
    /// <summary>
    /// Gets the point at time.
    /// </summary>
    /// <returns>
    /// The point at time.
    /// </returns>
    /// <param name='t'>
    /// T.
    /// </param>
    public Vector3 GetPointAtTime(float t)
    {
        return CreateBezierForPoint(t);
    }
    /// <summary>
    /// Creates the bezier for point.
    /// </summary>
    /// <returns>
    /// The bezier for point.
    /// </returns>
    /// <param name='t'>
    /// T.
    /// </param>
    private Vector3 CreateBezierForPoint(float t)
    {
        int x = (int) (t * (float)vectorArray.Length);

        if (x == vectorArray.Length)
        {
            x = 0;
        }

        // This is based from http://homepage.mac.com/nephilim/sw3ddev/bezier.html
        float newT = (t * (float)vectorArray.Length) - (float)x;
```

```

        Vector3 prev1 = vectorArray[CheckWithinArray(x, vectorArray.Length)];
        Vector3 this1 = vectorArray[CheckWithinArray(x + 1, vectorArray.Length)];
        Vector3 next1 = vectorArray[CheckWithinArray(x + 2, vectorArray.Length)];
        Vector3 far1 = vectorArray[CheckWithinArray(x + 3, vectorArray.Length)];

        Vector3 delta1 = (next1 - prev1) * .166f;
        Vector3 delta2 = (far1 - this1) * .166f;

        return DoBezierForNPoints(newT, new Vector3[] { this1, this1 + delta1, next1 -
delta2, next1 });
    }
    /// <summary>
    /// Checks the within array.
    /// </summary>
    /// <returns>
    /// The within array.
    /// </returns>
    /// <param name='x'>
    /// X.
    /// </param>
    /// <param name='c'>
    /// C.
    /// </param>
    private int CheckWithinArray(int x, int c)
    {
        if (x >= c)
        {
            return x % c;
        }
        else
        {
            return x;
        }
    }
    /// <summary>
    /// finds the bezier for N points.
    /// </summary>
    /// <returns>
    /// The bezier for N points.
    /// </returns>
    /// <param name='t'>
    /// T.
    /// </param>
    /// <param name='currentArray'>
    /// Current array.
    /// </param>
    private Vector3 DoBezierForNPoints(float t, Vector3[] currentArray)
    {
        Vector3 returnVector = Vector3.zero;

        float oneMinusT = (1f - t);

        int n = currentArray.Length - 1;

        for (int i = 0; i <= n; i++)
        {
            returnVector += BinomialCoefficient(n, i) * Mathf.Pow(oneMinusT, n - i) *
Mathf.Pow(t, i) * currentArray[i];
        }

        return returnVector;
    }
    #region Standard Maths methods
    /// <summary>

```

```

    /// Finds the Binomials coefficient.
    /// </summary>
    /// <returns>
    /// The coefficient.
    /// </returns>
    /// <param name='n'>
    /// N.
    /// </param>
    /// <param name='k'>
    /// K.
    /// </param>
    private float BinomialCoefficient(int n, int k)
    {
        if ((k < 0) || (k > n)) return 0;
        k = (k > n / 2) ? n - k : k;
        return (float) FallingPower(n, k) / Factorial(k);
    }
    /// <summary>
    /// Finds Factorial of specified n.
    /// </summary>
    /// <param name='n'>
    /// N.
    /// </param>
    private int Factorial(int n)
    {
        if (n == 0) return 1;
        int t = n;
        while (n-- > 2)
            t *= n;
        return t;
    }
    /// <summary>
    /// Falling power.
    /// returning an angle of fall
    /// </summary>
    /// <returns>
    ///
    /// The power.
    /// </returns>
    /// <param name='n'>
    /// N.
    /// </param>
    /// <param name='p'>
    /// P.
    /// </param>
    private int FallingPower(int n, int p)
    {
        int t = 1;
        for (int i = 0; i < p; i++) t *= n--;
        return t;
    }

    #endregion
}

```

ItemController.cs

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Item controller.
/// Gareth Wright 2013 - uses static accessibles to obtain score - this can then be
passed around but is dangerous
/// ..it should really be placed into a delegate state system i think -future dev point
THIS IS A FUTURE DEVELOPMENT
/// The class itself is designed to allow object activations to occur on gameobjects
dependent on a clause being met
/// In this case, the script has been personlaised to deliver areas and bones current
state updates.
/// </summary>
public class ItemController : MonoBehaviour
{
    /// <summary>
    /// The item count.
    /// </summary>
    public static int itemCount = 0; //available to all system
    /// <summary>
    /// The area count.
    /// </summary>
    public static int areaCount = 0; //available to all system
    /// <summary>
    /// The total number to collect.
    /// </summary>
    public int totalNumberToCollect = 9;
    /// <summary>
    /// The total areas to visit.
    /// </summary>
    public int totalAreasToVisit = 12;
    /// <summary>
    /// The story complete.
    /// </summary>
    public bool storyComplete = false; //completed bool
    /// <summary>
    /// The go.
    /// for teleporter pads
    /// </summary>
    public GameObject go = null;
    /// <summary>
    /// The go2.
    /// </summary>
    public GameObject go2 = null;
    /// <summary>
    /// The final go.
    /// </summary>
    public GameObject finalGo = null;
    /// <summary>
    /// The style.
    /// </summary>
    public GUIStyle style = null;
    /// <summary>
    /// The window text1.
    /// </summary>
    private string winText1 = " ";
    /// <summary>
    /// The window text2.
    /// </summary>
    private string winText2 = " ";
    /// <summary>
    /// The alpha.
```

```

    /// color cache alpha for game object
    /// </summary>
    private Color alpha;
    /// <summary>
    /// The lhs rect.
    /// screen counter-offset left
    /// </summary>
    private Rect lhsRect;
    /// <summary>
    /// The rhs rect.
    /// screen counter-offset right
    /// </summary>
    private Rect rhsRect;

    /// <summary>
    /// Start this instance.
    /// assess the gameobjects to see if null
    /// if no tnull set their active state ot false
    /// </summary>
    void Start()
    {
        /// set up right and left hand side rects for reference
        lhsRect = new Rect(Screen.width - Screen.width, Screen.height -
Screen.height, 201, 57);
        rhsRect = new Rect(Screen.width - 200, Screen.height - Screen.height, 201,
57);

        if(go != null)
        {
            /// cache an aplha value as colour
            alpha = new Color(1.0f, 1.0f, 1.0f, 0.0f);
            go.SetActive(false);
            return;
        }
        else
        {
            Debug.Log("NOT all GameObject have a reference added to them in the
ItemController script\n Do u need to add some gameobjects in your inspector ");
        }
        if(finalGo != null)
        {
            /// cache an aplha value as colour
            alpha = new Color(1.0f, 1.0f, 1.0f, 0.0f);
            finalGo.SetActive(false);
            return;
        }
        else
        {
            Debug.Log("NOT all GameObject have a reference added to them in the
ItemController script\n Do u need to add some gameobjects in your inspector ");
        }
    }
    /// <summary>
    /// Update this instance.
    /// conditionals to assess if itemCount or areaCount is achieved
    /// allowing gameobject to be set to active if conditional met
    /// </summary>
    void Update()
    {
        /// test
        if(Input.GetKeyDown (KeyCode.KeypadPlus))
        {
            itemCount++;
            areaCount++;
        }
        /// conditionals
        /// if not equal to specified total

```

```

        if(itemCount < totalNumberToCollect)
        {
            /// set child go to false
            go.SetActive(false);
            go2.SetActive(false);
        }
        else
        {
            /// set child go to false
            go.SetActive(true);
            go2.SetActive(true);

            Color itemAlpha = new Color(0.5f, 0.5f, 0.5f, 0.5f);

            go.renderer.material.color = itemAlpha;
        }
        /// if not equal to specified total
        if(areaCount <= totalAreasToVisit)
        {
            if(itemCount != totalNumberToCollect && areaCount !=
totalAreasToVisit)
            {
                /// set child go to false
                finalGo.SetActive (false);
            }
            else
            {
                /// set child go to false
                finalGo.SetActive (true);
            }
        }
    }/// end update

    /// <summary>
    /// Raises the GUI event.
    /// reassess validity of item and area counts concerning GUI visuals
    /// If achieved throw a mssage to the screen
    /// By adding 1 to each respective item the gui can be
    /// enabled based on this condition
    /// </summary>
    void OnGUI()
    {
        lhsRect = new Rect(Screen.width - Screen.width, Screen.height -
Screen.height, 201, 57);
        rhsRect = new Rect(Screen.width - 190, Screen.height - Screen.height, 201,
57);

        ///string to display in GUI box
        string itemText = "Total Bones " + itemCount;
        string areaText = "Total Areas Visited " + areaCount;
        string itemsToGetText = "Bones To collect " + (totalNumberToCollect -
itemCount);
        string areasToGetText = "Areas To collect " + (totalAreasToVisit -
areaCount);

        if(itemCount <= totalNumberToCollect)
        {
            /// tyle.font.material.color = Color.white;
            if(itemCount >= totalNumberToCollect)
            {
                winText1 = "Well Done\nYou Collected all " + itemCount + "
bones!!!" + "\n Return them to Kitty Jay`s Grave...\nIf you are ready...\n Look around
you for a portal to her resting place\n& return her bones!!!";
            }
        }
    }
}

```

```

GUI.Box (new Rect(Screen.width/2 - 250, (Screen.height/3), 500,
175), winText1, style);
// GUI.Box (new Rect(0, 15, 500, 175), winText1, style);
GUI.Label (new Rect(Screen.width - Screen.width/2,
Screen.height - Screen.height/3, 370, 65)," Remove, press 'R' on Keys or\n 'A' on
Controller " );
if(Input.GetButtonUp ("ClearGui"))
{
    // breach the itemcount and automatically remove the
relevant GUI
    itemCount++;
}
}
else
{
    GUI.BeginGroup(lhsRect,style);
    // position far left hand side
    GUI.Box (new Rect(0,15, 190, 20), itemsToGetText, style);
    // GUI.Box (new Rect(Screen.width - (Screen.width), 20, 170,
20), itemsToGetText, style);
    GUI.Box (new Rect(0, 30, 190, 20), itemText, style);
    GUI.EndGroup();
}
}
// this conditional checks against the visited areas , as each is
collected the count increments and is used for nested conditional inside
if(areaCount <= totalAreasToVisit)
{
    // if total reached
    if(areaCount >= totalAreasToVisit)
    {
        // assign text
        winText2 = "Well Done\nYou've seen all\t " + areaCount + "
areas" + "\n If you've already found all the bones..." + "\n& you feel you've seen all
you want to...\n walk into the tree by the car park\n to conclude your Hound Tor
experience";

        GUI.Box(new Rect(Screen.width/2 - 175, (Screen.height/3), 500,
350), winText2, style);
        // /GUI.Box(new Rect(Screen.width - 200, Screen.height -
Screen.height, 350, 80), winText2, style);

        GUI.Label (new Rect(Screen.width - Screen.width/2,
Screen.height - Screen.height/3, 370, 65)," Remove, press 'R' on Keys or\n 'A' on
Controller " );
        if(Input.GetButtonUp ("ClearGui"))
        {
            // breach the areacount and automatically remove the
relevant GUI
            areaCount++;
        }
    }
    else
    {
        GUI.BeginGroup(rhsRect, style);
        GUI.Label (new Rect(0, 15, 190, 20), areasToGetText, style);
        GUI.Label (new Rect(0, 30, 190, 20), areaText, style);
        GUI.EndGroup ();
    }
}
}
}
}

```

CollectItems.cs

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Collect items.
/// attached to a gameobject
/// leverages a single audio source
/// and waits a "time" before firing off
/// </summary>
public class CollectItems : MonoBehaviour
{
    /// <summary>
    /// The audio.
    /// audio cache
    /// </summary>
    public AudioSource audio;
    /// <summary>
    /// The time. holds an inspector assignable time value
    /// used in co routine
    /// </summary>
    public float time = 2.5f;
    /// <summary>
    /// Raises the trigger enter event.
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// Col must be equal to the "Player" tag
    /// </param>
    void OnTriggerEnter(Collider col)
    {
        switch(col.gameObject.tag)
        {
            case "Player":
                /// add item to static item count in ItemController
                ItemController.itemCount++;
                StartCoroutine (PlayAudio (time));
                /// remove object from game
                Destroy (this.gameObject);

                break;
        }
    }
    /// <summary>
    /// Plays the audio.
    /// sets looping to false
    /// controls doppler if 3d sound
    /// </summary>
    /// <returns>
    /// The audio.
    /// </returns>
    /// <param name='_time'>
    /// _time.
    /// </param>
    IEnumerator PlayAudio(float _time)
    {
        _time = time;
        audio.dopplerLevel = 0.0f;
        audio.loop = false;
        audio.Play();
        yield
            return
            new
```

```

        WaitForSeconds(_time);
    }
}

```

CollectAreas.cs

// Converted from UnityScript to C# at http://www.M2H.nl/files/js_to_c.php - by Mike Hergaarden
 // Do test the code! You usually need to change a few small bits.

```

using UnityEngine;
using System.Collections;
/// <summary>
/// Collect areas.
/// attached to a gameobject
/// leverages a single audio source
/// and waits a "time" before firing off
/// </summary>
public class CollectAreas : MonoBehaviour
{
    /// <summary>
    /// The audio.
    /// audio cache
    /// </summary>
    public AudioSource audio;
    /// <summary>
    /// The time. holds an inspector assignable time value
    /// used in co routine
    /// </summary>
    public float time = 2.5f;

    /// <summary>
    /// Raises the trigger enter event.
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// Col must be equal to the "Player" tag
    /// </param>
    void OnTriggerEnter(Collider col)
    {
        switch(col.gameObject.tag)
        {
            case "Player":
                /// add item to static item count in ItemController
                ItemController.areaCount++;

                StartCoroutine (PlayAudio (time));
                /// remove object from game
                Destroy (this.gameObject);

                break;
        }
    }
    /// <summary>
    /// Plays the audio.
    /// sets looping to false
    /// controls doppler if 3d sound
    /// </summary>
    /// <returns>
    /// The audio.
    /// </returns>
    /// <param name='_time'>
    /// _time.

```

```
/// </param>
IEnumerator PlayAudio(float _time)
{
    _time = time;
    audio.dopplerLevel = 0.0f;
    audio.loop = false;
    audio.Play();
    yield
        return
        new
        WaitForSeconds(_time);
}
}
```

TriggerZone.cs

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Trigger zone.
/// Generic singleton class that purely sets a game objects
/// active state upon collider being triggered
/// </summary>
public class TriggerZone : MonoBehaviour
{
    public GameObject go;
    //public GameObject go2;
    public GameObject msgObject;

    /// <summary>
    /// Start this instance.
    /// </summary>
    void Start()
    {
        go.gameObject.SetActive(false);
        msgObject.gameObject.guiText.enabled = false;
    }
    /// <summary>
    /// Raises the trigger enter event.
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// assess's the current colliding objects tag and actions
    /// upon result
    /// </param>
    void OnTriggerEnter(Collider col)
    {
        if(col.gameObject.tag == "Player")
        {
            go.gameObject.SetActive(true);
            msgObject.gameObject.guiText.enabled = true;
        }
    }
}
```

GenericTriggerScript.cs

```
/*GWright 2013*/
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
/// <summary>
/// Generic trigger script.
/// This script employs a list to hold gameobjects
/// they are then set to false based on their renderer (they must have a renderer
atatched!)
/// Based upon the evaluation of the colliding object
/// they set the renderer to true or false
/// Used to instantiate 3d text mesh renderers at area compasses
/// </summary>
public class GenericTriggerScript : MonoBehaviour
{
    public List<GameObject> gos;
    public GameObject target;
    public bool alphaMin = true;

    void Start()
    {
        foreach(GameObject go in gos)
        {
            go.gameObject.renderer.enabled = false;
        }
    }
    /// <summary>
    /// Raises the trigger enter event.
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// is collider tagged with "Player" ?
    /// </param>
    void OnTriggerEnter(Collider col)
    {
        if(col.gameObject.tag == "Player")
        {
            foreach(GameObject go in gos)
            {
                go.gameObject.renderer.enabled = true;
            }
        }
    }
    /// <summary>
    /// Raises the trigger stay event.
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// turns now enabled objects towasrd the Player transform by
    /// assessing the tag again
    /// </param>
    void OnTriggerStay(Collider col)
    {
        if(col.gameObject.tag == "Player")
        {
            foreach(GameObject go in gos)
            {
```

```

        /// utilising .rotation instead of LookAt() function as it
        proved to have a reversing consequence with 3d text meshes
        go.transform.rotation = target.transform.rotation;
    }
}
}
/// <summary>
/// Raises the trigger exit event.
/// </summary>
/// <param name='col'>
/// Col.
/// loops through each gameobject triggered by colliding object and removes them
/// </param>
void OnTriggerExit(Collider col)
{
    if(col.gameObject.tag == "Player" )
    {
        foreach(GameObject go in gos)
        {
            go.gameObject.renderer.enabled = false;
        }
    }
}
}
}

```

FindMyGPS

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
/// <summary>
/// Find my GP.
/// G Wright 2013
/// simply places the objects transform position into some arbitrary floats
/// Not a complete script really
/// more of a future development
/// </summary>
public class FindMyGPS : MonoBehaviour
{
    /// <summary>
    /// The go.
    /// </summary>
    public GameObject go; // marker object
    /// <summary>
    /// The myradius.
    /// </summary>
    public float myradius = 5.0f; // globe ball radius
    /// <summary>
    /// The mylatitude.
    /// </summary>
    public float mylatitude; // lat
    /// <summary>
    /// The mylongitude.
    /// </summary>
    public float mylongitude; // long
    /// <summary>
    /// The x position.
    /// </summary>
    private float xPos, yPos, zPos;
    /// <summary>
    /// The last position.
    /// </summary>
    private Vector3 lastPos;
    /// <summary>
    /// The object warning.
    /// </summary>
    private string objWarning = "No Gameobject found, please add in the inspector";
    /// <summary>
    /// The player position.
    /// </summary>
    public static Vector3 playerPos;
    /// <summary>
    /// Start this instance.
    /// set long and lat to transforms position x and z
    /// </summary>
    void Start ()
    {
        if(go == null)
        {
            Debug.LogWarning(objWarning);
        }
        else
        {
            mylongitude = go.transform.position.x;
            mylatitude = go.transform.position.z;
        }
        lastPos = go.transform.position;
    }
}
```

```

    /// <summary>
    /// Update this instance.
    /// </summary>
    void Update ()
    {
        mylatitude = go.transform.position.x; /// make long and lat register
gameobject position
        mylongitude = go.transform.position.z; /// leave yPosition of go for
now as we are not interested in 3d geolocation yet

        if(go.transform.position != lastPos)
        {
            xPos = go.transform.position.x; /// assign x
            zPos = go.transform.position.z; /// assign z

            yPos = go.transform.position.y; /// assign y anyway to ensure
you have the data atleast if needed

        }
    }
}

```

PauseEvent.cs

```
using UnityEngine;
using System.Collections;

public class PauseEvent : MonoBehaviour
{
    /// <summary>
    /// The target object.
    /// easier access to the gameobject's components,
    /// in this instance we want the camera
    /// </summary>
    public GameObject targetObj;
    ///are we paused
    /// <summary>
    /// The paused.
    /// set to false
    /// </summary>
    private bool paused = false;
    /// <summary>
    /// Start this instance.
    /// </summary>
    void Start ()
    {

        Screen.lockCursor = true;
        Screen.showCursor = false;
        /// set our cache object component to false
        /// this component is in the Main Camera Game Object
        targetObj.GetComponent<GUIMenuManager>().enabled = false;
    }

    /// Update is called once per frame
    void Update ()
    {

        /// this keybutton check is functioned through the input indentifer string
        for input keys.
        /// this allows for both controller and keyboard to take the same ...
        /// ...identifier but can fire events over multiple buttons
        if(Input.GetButtonDown("Pause")) // keyboard P or xbox Start button
        {
            ///check if p pressed to pause game
            if(!paused)
            {

                Screen.lockCursor = false;
                Screen.showCursor = true;
                paused=true;
                Time.timeScale = 0.0f;
                /// set component menu to true
                targetObj.GetComponent<GUIMenuManager>().enabled = true;
            }
            else /// when pressed again
            {
                Screen.lockCursor = true;
                Screen.showCursor = false;
                paused=false;
                Time.timeScale = 1.0f - Time.timeScale;
                /// set component menu to false
                targetObj.GetComponent<GUIMenuManager>().enabled = false;
                /// yaay a working menu in Unity !
            }
        }
    }
}
```


GUIMenuManager.cs

```
using UnityEngine;
using System;
using System.Collections;
using System.Collections.Generic;
/// <summary>
/// GUI menu manager. - called by Pause Event from accompanying script component
/// Gruffy2013
/// This is a delegate system to deal with changing gui elements at the start and
/// pause menu of my game
/// The delegate is passed in visa vis the instance delegate declared above it
/// This then applys the state passd in as the arguement
/// For each argument there is a method to carry out.
/// FUTURE DEVS - change to event handlers to create system wide gui controls to handle
/// all game states
/// - this means a state manager that could house these methods as class methods,
/// perhaps...
/// ...a base that can derive a more uniformed menu at the start
/// </summary>
public class GUIMenuManager : MonoBehaviour
{
    // cache some audio clips
    public AudioClip[] aClip;
    /// <summary>
    /// The style of choice.
    /// </summary>
    public GUIStyle styleOfChoice;
    /// <summary>
    /// The uniform skin.
    /// </summary>
    public GUISkin[] uniformSkin;
    /// <summary>
    /// GUI delegate.
    /// to pass states to
    /// </summary>
    private delegate void GUIDelegate();
    /// <summary>
    /// The current delegate.
    /// make instance of GUIDelegate
    /// </summary>
    private GUIDelegate curDelegate;
    /// <summary>
    /// The x position.
    /// base pos caches
    /// </summary>
    private int xPos, yPos;
    /// <summary>
    /// The next.
    /// bool to check if button sequence followed
    /// </summary>
    private bool next = false;
    /// <summary>
    /// The gamma.
    /// corrective value for gamma render setting in settings menu
    /// </summary>
    public float gamma;
    /// <summary>
    /// The fog distance.
    /// corerctive value for gamma render setting in settings menu
    /// </summary>
    public float fogDistance;
    /// <summary>
    /// Start this instance.
```

```

/// set current delegate to main menu method
/// </summary>
public void Start()
{
    this.curDelegate = MainMenu;
}

/// <summary>
/// Raises GUI event. (twice per frame)
/// </summary>
public void OnGUI()
{
    /// access rendersettings
    RenderSettings.ambientLight = new Color(gamma, gamma, gamma, 1.0f);
    RenderSettings.fogDensity = fogDistance;
    ///declare the current skin array option
    GUI.skin = uniformSkin[0];
    /// create gui layout, full screen estate
    GUILayout.BeginArea(new Rect(0,0,Screen.width, Screen.height ));
    GUILayout.BeginVertical();

    xPos = 10;
    yPos = 25;

    GUILayout.EndVertical();
    GUILayout.EndArea();
    /// apply initialised delegate state
    this.curDelegate();
}/// end GUI loop

/// <summary>
/// Main menu.
/// </summary>
//main menu methods
private void MainMenu()
{
    camera.backgroundColor = Color.white;
    /// check it
    Debug.Log ("main menu");
    GUI.skin = uniformSkin[1];
    /// button choices
    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height
/2) ,Screen.width,110), "Press 'P' or '\n'Start Button'\nto Return to Game",
styleOfChoice))
    {
        audio.clip = aClip[0];
        audio.Play();
    }
    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height /2)+
130 ,Screen.width,40), "Options", styleOfChoice))
    {
        audio.clip = aClip[1];
        audio.Play();
        /// options button clicked, switch to new menu
        this.curDelegate = OptionsMenu;
    }
    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height /2) +
180 ,Screen.width,40), "Exit", styleOfChoice))
    {
        audio.clip = aClip[1];
        audio.Play();
        /// options button clicked, switch to new menu
        this.curDelegate = ExitMenu;
    }
}
}

```

```

/// Options menu.
//button Methods
private void OptionsMenu()
{
    /// audio.PlayOneShot(changeClip);
    camera.backgroundColor = Color.grey;
    /// check it
    Debug.Log ("options menu");

    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height
/2) ,Screen.width,40), "Return To Main", styleOfChoice))
    {
        audio.clip = aClip[0];
        audio.Play();
        /// options button clicked, switch to new menu
        this.curDelegate = MainMenu;
    }
    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height /2) +
130 ,Screen.width,40), "Settings", styleOfChoice))
    {
        audio.clip = aClip[1];
        audio.Play();
        /// options button clicked, switch to new menu
        this.curDelegate = SettingsMenu;
    }
    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height /2) +
180 ,Screen.width,40), "Controls", styleOfChoice))
    {
        audio.clip = aClip[1];
        audio.Play();
        /// options button clicked, switch to new menu
        this.curDelegate = ControlsMenu;
    }
}
/// <summary>
/// Settings menu.
/// </summary>
//settings menu methods
private void SettingsMenu()
{
    camera.backgroundColor = Color.cyan;
    ///check it
    Debug.Log ("Settings menu");
    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height
/2) ,Screen.width,40), "Return To Main", styleOfChoice))
    {
        audio.clip = aClip[0];
        audio.Play();
        this.curDelegate = MainMenu;
    }
    /// gamma = GUI.HorizontalSlider( new Rect((Screen.width / 2),(Screen.height
/2 + 50) ,Screen.width - (Screen.width /2),40));
    /// gamma adjust slider
    GUI.Label (new Rect ((Screen.width / 2),(Screen.height /2 + 85) ,200, 40),
"Gamma", styleOfChoice);
    gamma = GUI.HorizontalSlider(new Rect ((Screen.width / 3),(Screen.height /2
+ 130) ,(Screen.width - Screen.width/2), 40), gamma, 0, 1.0f);
    /// fog adjust slider
    GUI.Label (new Rect ((Screen.width / 2),(Screen.height /2 + 135),200, 40),
"Fog Distance", styleOfChoice);
    fogDistance = GUI.HorizontalSlider(new Rect ((Screen.width /
3),(Screen.height /2 + 180) ,(Screen.width - Screen.width/2), 40), fogDistance, 0,
0.03f);
}

```

```

/// <summary>
/// Controls menu- to show user inputs on the screen
/// </summary>
private void ControlsMenu()
{
    camera.backgroundColor = Color.cyan;
    /// check it
    Debug.Log ("Controls menu");
    if (GUI.Button (new Rect ((Screen.width / 2 - 20),Screen.height -
40 ,Screen.width,40), "Return To Main", styleOfChoice))
    {
        audio.clip = aClip[0];
        audio.Play();
        this.curDelegate = MainMenu;
    }
    GUI.BeginGroup (new Rect(Screen.width /2 - 275, 0, 550 , 670),"CONTROLS",
styleOfChoice);
    /// control scheme
    GUI.Label (new Rect (10,52,540,40), " Move Forward:\n W (ARROW UP KEY)\n
or Left Analogue ThumbStick", styleOfChoice);
    GUI.Label (new Rect (10,152,540,40), " Move Backward:\n S (ARROW DOWN
KEY)\n or Left Analogue ThumbStick", styleOfChoice);
    GUI.Label (new Rect (10,252,540,40), " Move Left:\n A / (ARROW_LEFT KEY)\n
or Left Analogue ThumbStick", styleOfChoice);
    GUI.Label (new Rect (10,352,540,40), " Move Right:\n D / (ARROW_RIGHT
KEY)\n or Left Analogue ThumbStick", styleOfChoice);
    GUI.Label (new Rect (10,452,540,40), " Look: Mouse\n or Right Analogue
ThumbStick", styleOfChoice);
    GUI.Label (new Rect (10,552,540,40), " Pause: KeyBoard P\n or Controller
START button", styleOfChoice);
    GUI.EndGroup(); }
/// <summary>
/// Exit menu.
/// exit menu asks if sure then presents a further confirmation GUI
/// After confirmation, application exits
/// </summary>

private void ExitMenu()
{
    camera.backgroundColor = Color.yellow;
    Debug.Log ("exit menu");
    if (GUI.Button (new Rect ((Screen.width / 2),(Screen.height
/2) ,Screen.width,40), "Click to Exit Game", styleOfChoice))
    {
        audio.clip = aClip[1];
        audio.Play();

        next = true;
    }
    if(next)
    {
        camera.backgroundColor = Color.red;
        GUI.Box (new Rect ((Screen.width / 2),(Screen.height /2)+
50 ,Screen.width,40), "SURE", styleOfChoice);

        if(GUI.Button (new Rect ((Screen.width / 2),(Screen.height /2)+
110 ,Screen.width,40), "Yes ?", styleOfChoice))
        {
            Application.Quit();
        }/// options button clicked, switch to new menu
        if(GUI.Button (new Rect ((Screen.width / 2),(Screen.height /2)+
170 ,Screen.width,40), "No ?", styleOfChoice))
        {
            audio.clip = aClip[0];
            audio.Play();
            this.curDelegate = MainMenu;
        }
    }
}

```

```
}}}}
```

GUIHoverControls

```
using UnityEngine;
using System.Collections;
/// <summary>
/// GUI hover controls.
/// Forces an audio source to the instance
/// plays an audio object on mouse up
/// changes colour on mouse over and exit
/// loads or plays based on textmesh being clicked on the screen
/// </summary>
[RequireComponent(typeof(AudioSource))]
public class GUIHoverControls : MonoBehaviour
{
    public int levelToLoad;
    public AudioClip narrative;
    public AudioClip beep;
    public bool isQuitButton = false;
    public float wait = 5.0f;
    public Material mat1;
    public Material mat2;
    public Material mat3;
    public Material fadedOutMat;

    public TextMesh txtMesh;
    // Use this for initialization
    private bool isClicked = false;
    /// <summary>
    /// Raises the mouse over event.
    /// </summary>
    void OnMouseOver()
    {
        /// change the material when hovering over
        ChangeColor(mat2);

        /// if leftmouse button cllicked
        if(Input.GetMouseButtonUp(0))
        {
            /// change mat to signify click occurence
            ChangeColor(mat3);
            /// if this is a wuit button
            if(isQuitButton)
            {
                /// audio.clip = clickBeep;
                audio.PlayOneShot(beep);

                /// exit game
                Application.Quit();
            }
            else /// this is a button to load a level
            {
                Debug.Log ("loading level in 5 secs");
                /// specifics hack to check if button is start game to allow
                button audio to yield before starting the level
                if(txtMesh.text == "Hound Tor")
                {
                    /// GUI.Label (new Rect(Screen.width/2,Screen.height
                    /3,100,40), "Click to for Intro");
                    isClicked = true;
                    Debug.Log (isClicked);
                    StartCoroutine(StartIntroTalk());
                }
            }
        }
    }
}
```

```

        else if(txtMesh.text == "Play gamE")
        {
            isClicked = true;
            Debug.Log (isClicked);
            StartCoroutine(LoadALevel());
        }
        else /// we checked and it wasnt the start button
        {
            isClicked = false;
            Debug.Log (isClicked);
            /// StartCoroutine(LoadChosenLevel());
        }
    }
}

/// <summary>
/// Raises the mouse exit event.
/// when the mouse exits the textmesh's collision area
/// </summary>
void OnMouseExit()
{
    /// return colour to normal white faded alpha
    ChangeColor (mat1);
}

/// <summary>
/// Changes the color.
/// change colour method taking a material in as the argument
/// </summary>
/// <param name='mat'>
/// Mat.
/// </param>/ <summary>
/// Changes the color.
/// </summary>
/// <param name='mat'>
/// Mat.
/// </param>/ <summary>
/// Changes the color.
/// </summary>
/// <param name='mat'>
/// Mat.
/// </param>///
void ChangeColor(Material mat)
{
    /// if not mat1 or mat 2
    if(mat != fadedOutMat && mat != mat2 && mat != mat3)
    {
        /// mat must be mat 1
        mat = mat1;
    }
    else if(mat != fadedOutMat && mat != mat1 && mat!= mat3)
    {
        /// mat must be mat 2
        mat = mat2;
    }
    else if(mat != fadedOutMat && mat!= mat1 && mat!= mat2)
    {
        /// mat must be mat 3
        mat = mat3;
    }
    else
    {
        /// mat must be mat 4
        mat = fadedOutMat;
    }
}

```

```

    }
    /// assign new material to the textMesh being operated on
    txtMesh.renderer.material = mat;
}
/// <summary>
/// Starts a narrative.
///
/// </summary>
/// <returns>
/// The intro talk after time
/// </returns>
IEnumerator StartIntroTalk()
{
    audio.clip = narrative;
    audio.Play();
    if(isClicked)
    {
        ChangeColor (fadedOutMat);
    }
    yield
    return
        new WaitForSeconds(wait);
}
/// <summary>
/// Loads a level based on a mouse input to start game
/// </summary>
/// <returns>
/// The A level.
/// </returns>
IEnumerator LoadALevel()
{
    audio.PlayOneShot(beep);
    if(isClicked)
    {
        ChangeColor (fadedOutMat); /// txtMesh.renderer.material.color.a =
new
Color(txtMesh.renderer.material.color.r,txtMesh.renderer.color.g,txtMesh.renderer.color.
b,0.0f);
    }
    yield
    return
        new WaitForSeconds(wait);
        Application.LoadLevel(levelToLoad);
}

}/// end class

```

ExitBackUp.cs

```

using UnityEngine;
using System.Collections;
/// Exit back up.
public class ExitBackUp : MonoBehaviour
{
    /// <summary>
    /// Update this instance.
    /// </summary>
    void Update ()
    {
        if(Input.GetKeyUp(KeyCode.Escape))
        {
            Application.Quit();
        }
    }
}

```

```

    }
}}

```

EndCreditsScript.cs

```

using UnityEngine;
using System.Collections;
public class EndCreditsScript : MonoBehaviour
{
    void OnTriggerEnter(Collider col)
    {
        if(col.gameObject.tag == "Player")

        {
            Application.LoadLevel (2); /// end credits scene
        }
    }
}

```

GUIHelper.cs

```

using UnityEngine;
using System.Collections;
/// <summary>
/// GUI helper.
/// Help and extension methods to be placed in here
/// </summary>
public class GUIHelper : MonoBehaviour
{
    public static Rect screenRect(float screenPosx,float screenPosy,float screenW,
float screenH)
    {
        float xPos = screenPosx * Screen.width;
        float yPos = screenPosy * Screen.height;
        float sW = screenW * Screen.width;
        float sH = screenH * Screen.height;

        return new Rect(xPos,yPos,sW,sH);
    }
}

```

CleanUp.cs

```

using UnityEngine;
using System.Collections;
/// <summary>
/// Clean up.
/// destroy object after time passed
/// </summary>
public class CleanUp : MonoBehaviour
{
    public float timeToLive = 5.0f;
    /// <summary>
    /// Start this instance.
    /// </summary>
    void Start ()
    {
        Destroy(gameObject, timeToLive); /// Destroy(obj/component to remove,
optional time delay length)
    }
}

```

RotateObject

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Rotate object.
/// Created for convenience by G Wright 2013 (gruffy.wright@gmail.com)
/// This script can be attached to any game object and give the developer simplified
access to GameObject rotations
/// The Vector3 type's built-in scope operators are...
/// .up = facing the GameObject
yields an anticlockwise rotation on Y axis
/// .down = facing the GameObject yields a
clockwise rotation on Y axis
/// .left = facing the GameObject yields an
anticlockwise rotation on X axis
/// .right = facing the GameObject yields a
clockwise rotation on X axis
/// .back = facing the GameObject yields an
anticlockwise rotation on Z axis
/// .forward = facing the GameObject yields a
clockwise rotation on Z axis
///
/// Below are Bespoke, as I could not find built in operations in Unity to deal with
all axes rotations...
///
/// .all axes rotate positively = facing the GameObject yields an anticlockwise
rotation on all axes
/// .all axes rotate negatively = facing the GameObject yields a clockwise
rotation on all axes
///
/// </summary>
public class RotateObject : MonoBehaviour
{
    public float chosenSpeed; //multiplier
    public float[] variants;
    //inspector accessed enumerator type
    public enum AxisRotationChoice {AntiClockWiseY, ClockWiseY, AntiClockWiseX,
ClockWiseX, AntiClockWiseZ, ClockWiseZ, AntiClockWiseMix, ClockWiseMix};
    //script based cache to operate on
    public AxisRotationChoice axisRotationChoice;
    private AxisRotationChoice axisRotChoice;

    /// <summary>
    /// Start this instance.
    /// sets the private cache to equal the enumeration choice
    /// </summary>
    void Start()
    {
        axisRotChoice = axisRotationChoice; /// cache for inner operations
    }

    /// <summary>
    /// Update this instance.
    /// By selection in the inspector through the enumeration
    /// this script just simplifies the process of getting the right rotation on an
object
    /// </summary>
    void Update ()
    {
        /// cache the public enum to the private enum instance for safe operations
        axisRotChoice = axisRotationChoice;
```

```

        switch(axisRotChoice) // enumerator action based on inspector
selected choice - changes are updated at runtime too for editor debugging
        {
            case AxisRotationChoice.AntiClockWiseY
:
transform.Rotate(Vector3.up * Time.deltaTime * chosenSpeed); /// rotate
anticlockwise on Y

            break;
            case AxisRotationChoice.ClockWiseY
:
transform.Rotate(Vector3.down * Time.deltaTime * chosenSpeed); /// rotate clockwise
on Y

            break;
            case AxisRotationChoice.AntiClockWiseX
:
transform.Rotate(Vector3.left * Time.deltaTime * chosenSpeed); /// rotate
anticlockwise on X

            break;
            case AxisRotationChoice.ClockWiseX
:
transform.Rotate(Vector3.right * Time.deltaTime * chosenSpeed); /// rotate
clockwise on X

            break;
            case AxisRotationChoice.AntiClockWiseZ
:
transform.Rotate(Vector3.forward * Time.deltaTime * chosenSpeed); /// rotate
naticlockwise on Z

            break;
            case AxisRotationChoice.ClockWiseZ
:
transform.Rotate(Vector3.back * Time.deltaTime * chosenSpeed); /// rotate clockwise
on Z

            break;
            case AxisRotationChoice.AntiClockWiseMix
:
transform.Rotate(Time.deltaTime * chosenSpeed, Time.deltaTime * chosenSpeed,
Time.deltaTime * chosenSpeed); /// rotate naticlockwise on Z

            break;
            case AxisRotationChoice.ClockWiseMix
:

```

```

    transform.Rotate(Time.deltaTime *- chosenSpeed, Time.deltaTime *- (chosenSpeed -
10), Time.deltaTime *- (chosenSpeed + 10)); /// rotate anticlockwise on Z

    /// Some hardcoding has been applied in the parameter`s arguments below just to
vary the rotation of each axis, as they all spin together simultaneously

    break;

    default
: /// Here we could default to a rotation by removing a case option from above and
just adding the functional bit...

    /// "transform.Rotate(of your choice)" into this section

    /// and removing/ commenting out this operation below.

    /// Do not rotate as a default - never achieved but a safe ending and good practice
- though h

    transform.Rotate(0.0f, 0.0f, 0.0f);

    break;
}
}
}

```

TriggerObject.cs

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
/// <summary>
/// Trigger object.
/// using on trigger function to change material, guitext.
/// coordinates can be offset with coords
/// </summary>
public class TriggerObject : MonoBehaviour
{
    /// <summary>
    /// The object to change.
    /// </summary>
    public GameObject objectToChange;
    /// <summary>
    /// The original material.
    /// </summary>
    public Material originalMaterial;
    /// <summary>
    /// The change to material.
    /// </summary>
    public Material changeToMaterial;
    /// <summary>
    /// The trigger field is.
    /// </summary>
    public bool triggerFieldIs = false;
    /// <summary>
    /// The style.
    /// </summary>
    public GUIStyle style;
    /// <summary>
    ///
    /// </summary>
    public List<string> content;
    /// <summary>
    /// The distance.
    /// </summary>
    public int distance = -1;
    /// <summary>
    /// The coords.
    /// </summary>
    public List<Vector2> coords;
    /// <summary>
    /// The go.
    /// </summary>
    private GameObject go;
    /// <summary>
    /// Start this instance.
    /// sets current instance object to the changed object
    /// sets renderer on new object to equal a transparent material
    /// </summary>
    void Start()
    {
        go = objectToChange;
        go.renderer.material = originalMaterial;
    }
    /// <summary>
    /// Raises the trigger enter event.
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// </param>
```

```

void OnTriggerEnter(Collider col)
{
    if(col.gameObject.tag == "Player")
    {
        triggerFieldIs = true; /// triggerFieldIs true
        Debug.Log("trigger area entered ");
        go.renderer.material = changeToMaterial; /// set current object
renderer material
    }
}
/// <summary>
/// Raises the trigger exit event.
/// </summary>
/// <param name='col'>
/// Col.
/// </param>
void OnTriggerExit(Collider col)
{
    if(col.gameObject.tag == "Player")
    {
        triggerFieldIs = false; /// set triggerFieldIs to false
        /// go.renderer.material = originalMaterial;
        Debug.Log(" player left trigger area ");
    }
}
/// <summary>
/// Raises the GU event.
/// </summary>
void OnGUI()
{
    if (triggerFieldIs)
    {
        foreach(string c in content)
        {
            foreach(Vector2 v in coords)
            {
                GUI.Box(new Rect(v.x, v.y, 0.0f , 0.0f), c, style);
// create new gui box
            }
        }
    }
}

```

TriggerKittyAudio.cs

```

using System.Collections;
using UnityEngine;
///
/// <summary>
/// Trigger kitty audio class
/// This class defines the operations performed on audio
/// It is triggered by a change in the item count static cache for the total items
collected
/// For each increment in item an audio narrative will be triggered
/// The class is set to check if audio isplaying and not to override with the next slot
/// however - it does not hold that audio in queue to play after the last had ended.
/// This is a future refinement and does not
/// </summary>
public class TriggerKittyAudio : MonoBehaviour
{
    public AudioClip[] audioClip;
    public int item;
    private int prevItem = 0;
    private bool canPlay = true;
    /// <summary>

```

```

    /// Update this instance.
    /// assigns item controller item count to items and
    /// evaluates it against the current audio slot in the array
    /// then plays the ,atching element
    /// </summary>
    void Update ()
    {
        item = ItemController.itemCount;
        if (canPlay && item != prevItem)
            PlayCurrentClip();
    }
    /// <summary>
    /// sets the current clip.
    /// </summary>
    private void PlayCurrentClip()
    {
        if (item >= audioClip.Length) /// make sure your index isn't longer than the array
            item = audioClip.Length - 1;
        else if (item < 0)
        {
            canPlay = false;
            return;
        }
        /// if the item was already played, or the player is currently playing, or the
        /// audioclip hasn't been set, exit the function and do not execute any following code
        if (item == prevItem || audioClip[item] == null)
            return;

        StartCoroutine(PlaySound());
    }

    /// <summary>
    /// Play the sound.
    /// </summary>
    /// <returns>
    /// The current audio clip is played until finished
    /// the co routine then returns the play state to true
    /// </returns>
    private IEnumerator PlaySound()
    {
        canPlay = false;
        audio.clip = audioClip[item];
        audio.Play();
        prevItem = item;

        yield return new WaitForSeconds(3.0f);
        canPlay = true;
    }
}

```

Scroller2.cs

```
using UnityEngine;
using System.Collections;
/// <summary>
/// Scroller2.
/// This is an improved version of a previous script built to scroll the text as a gui
element.
/// It is used to display both the first in-game instructions and the games end credits
and acknowledgements
/// </summary>
public class Scroller2 : MonoBehaviour
{
    public string[] scrollLines;
    public GUIStyle style;
    public float offset = 0.0f;      ///start position on 2d y
    public float speed = 100.0f;
    /// <summary>
    /// Raises the GUI event.
    /// positions text label to middle of screen and releases
    /// each line based on the rectangle offset amount, this is also used to
    /// stores an alpha value and replaces the labekl alpha with 0.0
    /// then over the calls to OnGUI reduces it for every line in the array
    created
    /// </summary>
    private void OnGUI()
    {
        offset += Time.deltaTime * speed;
        for (int i = 0; i < scrollLines.Length; i++)
        {
            float rectOffset = (scrollLines.Length*-20) + (i*25 + offset);

            float alpha =
Mathf.Sin((rectOffset/Screen.height)*180*Mathf.Deg2Rad);
            GUI.color = new Color(1,1,1, alpha);

            GUI.Label(new Rect((Screen.width - Screen.width /2) ,rectOffset ,64,
64),scrollLines[i], style);

            GUI.color = new Color(1,1,1,1);
        }
    }
}
```

Teleporter.cs

```
using UnityEngine;
/// <summary>
/// Teleporter.
/// </summary>
public class Teleporter : MonoBehaviour
{
    /// <summary>
    /// The destination.
    /// create instance of teleporter
    /// </summary>
    public Teleporter destination;
    /// <summary>
    /// The teleported.
    /// set flag to false
    /// </summary>
    private bool teleported = false;
    /// <summary>
    /// Start this instance.
    /// </summary>
    void Start()
    {
        if(destination != null)
        {
        }
        else
        {
            Debug.Log ("@ Teleport Area: THIS script requires a script type to be
assigned in inspector");
        }
    }
    /// <summary>
    /// Raises the trigger enter event.
    /// send the player to another world space destination
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// </param>
    void OnTriggerEnter(Collider col)
    {
        if (col.CompareTag("Player"))
        {
            if (!teleported)
            {
                destination.teleported = true; /// change teleported flag by accessing
the scripts instance
                col.gameObject.transform.position =
destination.gameObject.transform.position;
            }
        }
    }
    /// <summary>
    /// Raises the trigger exit event.
    /// when exiting a collider the flag is set ot false
    /// signalling end of routine and return to original state
    /// </summary>
    /// <param name='col'>
    /// Col.
    /// </param>
    void OnTriggerExit(Collider col)
    {
        if (col.CompareTag("Player"))
```

```

        {
            teleported = false; /// return to original flag state
        }
    }
}

```

ObjectStateChange.cs

```

using UnityEngine;
using System.Collections;
/// <summary>
/// Object state change./// </summary>
public class ObjectStateChange : MonoBehaviour
{
    public GameObject[] go;
    /// <summary>
    /// State the specified state.
    /// hough void , we want the method to operate on a parameter set outside of the
class
    /// this is achieved by the SendMessage Class functionality in Unity allowing us to
pass the argument to the parameter for operation easily
    /// An optimized route may be to set a delegate system up to pass the argument via
an event handler. this is a full system upgrade as part of future developments.
    /// as such , this script and paradigm of its use conforms to the Unity way and is
not used more than once so is not a heavy load...
    /// ...this coupled with managed garbage collection makes this script a viable
option
    /// method to handle setting game object to active or not
    /// </summary>
    /// <param name='state'>
    /// State.
    /// </param>
    void State( bool state)
    {/// cache bool
        bool _state = state;
        foreach(GameObject _go in go)
        {
            gameObject.renderer.enabled = _state; /// SetActive(_state);
        }
    }
}
}

```

RB_FPS_Gruffy.cs

```
/// <summary>
/// Character controls - Copyrights Original Code - Unify Community 2012/ Changed Code
- Gruffy 2013
/// These have been adapted from the Rigidbody FPS Walker code found at
/// http://wiki.unity3d.com/index.php?title=RigidbodyFPSWalker
/// The script, originally Javascript was adapted to C# and allows the player controll
system to be affected by real physics setup in the game
/// This is due to the character controller , including movement and mouse control
scripts, creating its own physics during runtime to move the player object about or
affect it against the game world
/// if needed.
/// Unify's example idea is based on games that may require workld physics objects to
interact with the player object as a result of a physics calculation and as a physics
based reaction, rather than inaccurately calculating this at incident.
/// overaall, we can start to see why this is a better option as we are establishing
the physics boundaries prior to runtime with rigidbody alone, with char controller it
happens in update (unstable for physics)
/// </summary>
///
using UnityEngine;
using System.Collections;

//important componenets to instance the object with minimum requirements
[RequireComponent (typeof (Rigidbody))]
[RequireComponent (typeof (CapsuleCollider))]
/// <summary>
/// R b_ FP s_ gruffy.
/// </summary>
public class RB_FPS_Gruffy : MonoBehaviour {

    public AudioClip audioClip;
    public float walkSpeed = 8.0f; //multiplier
    public float walkBackwardSpeed = 4.0f;
    public float sidestepSpeed = 8.0f;
    public float gravity = 10.0f;
    public float maxVelocityChange = 10.0f;
    public float inAirControl = 0.1f;
    public bool canJump = true;
    public float jumpHeight = 2.0f;
    public float slidingThreshold = 0.6f;
    public float slideMagnitudeConstraint = 0.4f;
    // added for run (set to Fire2. Change in Project Settings / Input if different
desired.)
    public bool canRun = true;
    public float runSpeed = 14.0f; // negative values here makes game unhappy
    public float runBackwardSpeed = 6.0f; // negative values here makes game unhappy
    // var runSpeedChange = 4.0; // negative values here makes game unhappy
    //vars to add sidestep functionality later on, if needed
    public bool canRunSidestep = true;
    public float runSidestepSpeed = 12.0f;
    //remember the touches for future dev
    public bool isTouchEnabled = false; //unused as yet

    private bool grounded = false;
    private Vector3 slideDir;
    private Vector3 velocity ;
    private Vector3 velocityChange;
    private Vector3 targetVelocity;
    //establish before game script is loaded that
    /// <summary>
    /// Awake this instance.

```

```

    /// </summary>
    void Awake ()
    {
        //our rotation is set to our z forward direction upon creation
        rigidbody.freezeRotation = true;
        //tell rigidbody to ignore game based gravity calculations (essential to a
        character controller )
        rigidbody.useGravity = false;
        //why?, because we will use forces to push the player object and this will
        include the Y axis (relative up and down in the world)
    }
    /// <summary>
    /// Fixed the update.
    /// </summary>
    void FixedUpdate ()
    {
        if (grounded) //true
        {

            // Calculate how fast to move based on inspector refined values
            targetVelocity = new Vector3(Input.GetAxis("Horizontal"), 0,
            Input.GetAxis("Vertical"));
            targetVelocity = transform.TransformDirection(targetVelocity);
            targetVelocity *= walkSpeed;

            // Apply a force that attempts to reach our target velocity
            velocity = rigidbody.velocity;
            velocityChange = (targetVelocity - velocity);

            velocityChange.x = Mathf.Clamp(velocityChange.x, -maxVelocityChange,
            maxVelocityChange);
            velocityChange.z = Mathf.Clamp(velocityChange.z, -maxVelocityChange,
            maxVelocityChange);
            velocityChange.y = 0;
            rigidbody.AddForce(velocityChange, ForceMode.VelocityChange);

            // Jump - not used in this game
            if (canJump && Input.GetButton("Jump"))
            {
                rigidbody.velocity = new Vector3(velocity.x,
                CalculateJumpVerticalSpeed(), velocity.z);
            }

            ApplyClimbConstraints();
        }

        //apply gravity manually for better tuning control
        rigidbody.AddForce(new Vector3 (0, -gravity * rigidbody.mass, 0));

        grounded = false;
    }

    /// <summary>
    /// Raises the collision stay event.
    /// </summary>
    void OnCollisionStay ()
    {
        grounded = true;
    }

    /// <summary>
    /// Calculates the jump vertical speed.
    /// </summary>
    /// <returns>
    /// The jump vertical speed.
    /// </returns>
    float CalculateJumpVerticalSpeed ()

```

```

{
    // From the jump height and gravity we deduce the upwards speed
    // for the character to reach at the apex.
    return Mathf.Sqrt(2 * jumpHeight * gravity);
}

/// <summary>
/// Applies the climb constraints.
/// using raycast downwards to determine angle of slope
/// not fully implememnted yet but applying a slide constraint to a
/// rigidbody method below
/// </summary>
void ApplyClimbConstraints()
{
    if (grounded)
    {
        return;
    }

    slideDir = Vector3.down;
    RaycastHit hit;
    // ray origin is placed 1 unit up from player (0, 1, 0)
    // , ray direction facing down (0, -1, 0)
    //, cache data to hit variable for use
    //If the transforms y were to start at 0 we would add "+ Vector3.up" to...
    //..the Raycast's 1st argument to ensure its at least 1 unit from the ground.
    //The default transform that is prefabbed is 2 m tall and the transforms y..
    //..scale is set to 0.835 so it never breaches 1 for this argument to be..
    //..valid as a raycast parameter.

    if(Physics.Raycast(transform.position , Vector3.down, out hit ))
    {
        if(hit.normal.y < slidingThreshold)
        {
            // argue slide direction with cached data from hit
            slideDir = new Vector3(hit.normal.x, -hit.normal.y, hit.normal.z);
        }
    }
    // the current magnitude is less than the magnitude constraint argument
    if (slideDir.magnitude < slideMagnitudeConstraint)
    {
        //add moveVects direction with slideDir direction
        //moveVect += slideDir;
        rigidbody.AddForce(velocityChange.x, velocityChange.y,
velocityChange.z, ForceMode.VelocityChange);
    }
    else // is > or ==
    {
        //can only slide - no input controls
        //moveVect = slideDir;
        rigidbody.AddForce(-velocityChange.x, -velocityChange.y, -
velocityChange.z, ForceMode.VelocityChange);
    }
}
}

```

MouseLook.cs

This class was originally leveraged from the unity MouseLook.cs class

```
using UnityEngine;
using System.Collections;

/// MouseLook rotates the transform based on the mouse delta.
/// Minimum and Maximum values can be used to constrain the possible rotation
/// To make an FPS style character:
/// - Create a capsule.
/// - Add a rigid body to the capsule
/// - Add the MouseLook script to the capsule.
/// -> Set the mouse look to use LookX. (You want to only turn character but not tilt it)
/// - Add FPSWalker script to the capsule
/// - Create a camera. Make the camera a child of the capsule. Reset it's transform.
/// - Add a MouseLook script to the camera.
/// -> Set the mouse look to use LookY. (You want the camera to tilt up and down like a head. The character already turns.)
[AddComponentMenu("Camera-Control/Mouse Look")]
public class MouseLook : MonoBehaviour {

    public enum RotationAxes { MouseXAndY = 0, MouseX = 1, MouseY = 2 }
    public RotationAxes axes = RotationAxes.MouseXAndY;
    public float sensitivityX = 15F;
    public float sensitivityY = 15F;

    public float minimumX = -360F;
    public float maximumX = 360F;

    public float minimumY = -60F;
    public float maximumY = 60F;

    float rotationX = 0F;
    float rotationY = 0F;

    Quaternion originalRotation;

    void Update ()
    {
        if (axes == RotationAxes.MouseXAndY)
        {
            // Read the mouse input axis
            rotationX += Input.GetAxis("Mouse X") * sensitivityX;
            rotationY += Input.GetAxis("Mouse Y") * sensitivityY;

            rotationX = ClampAngle (rotationX, minimumX, maximumX);
            rotationY = ClampAngle (rotationY, minimumY, maximumY);

            Quaternion xQuaternion = Quaternion.AxisAngle (Vector3.up,
Mathf.Deg2Rad * rotationX);
            Quaternion yQuaternion = Quaternion.AxisAngle (Vector3.left,
Mathf.Deg2Rad * rotationY);

            transform.localRotation = originalRotation * xQuaternion *
yQuaternion;
        }
        else if (axes == RotationAxes.MouseX)
        {
            rotationX += Input.GetAxis("Mouse X") * sensitivityX;
            rotationX = ClampAngle (rotationX, minimumX, maximumX);
```

```

        Quaternion xQuaternion = Quaternion.AxisAngle (Vector3.up,
Mathf.Deg2Rad * rotationX);
        transform.localRotation = originalRotation * xQuaternion;
    }
    else
    {
        rotationY += Input.GetAxis("Mouse Y") * sensitivityY ;
        rotationY = ClampAngle (rotationY, minimumY, maximumY);

        Quaternion yQuaternion = Quaternion.AxisAngle (Vector3.left,
Mathf.Deg2Rad * rotationY);
        transform.localRotation = originalRotation * yQuaternion;
    }

//      //give a debug and build escape to cursor option - gruffy2013
//      if(!Input.GetKeyUp(KeyCode.Escape))
//      {
//          Screen.lockCursor = true;
//          Screen.showCursor = false;
//      }
//      else
//      {
//          Screen.lockCursor = false;
//          Screen.showCursor = true;
//      }
}

void Start ()
{
    // Make the rigid body not change rotation
    if (rigidbody)
        rigidbody.freezeRotation = true;
    originalRotation = transform.localRotation;
    //lock the mouse to the screen
    //Screen.lockCursor = true;
    //Screen.showCursor = false;

}

public static float ClampAngle (float angle, float min, float max)
{
    if (angle < -360F)
        angle += 360F;
    if (angle > 360F)
        angle -= 360F;

    return Mathf.Clamp (angle, min, max);
}
}

```

Leveraged JavaScript's from the Purchased Particle game object

rampDownParticle.js

```
activateWithDelay.js × destroyThisTimed.js × ParticleScale.js × PrefabGenerator.js* × rampDownParticle.js* ×
rampDownParticle ▶ delayTime
1  var delayTime:float=0;
2  var delayPlusTime:float=0;
3  var rampDownTime:float=1;
4  var origMinEmission:float;
5  var origMaxEmission:float;
6
7  function Start () {
8
9  origMinEmission=particleEmitter.minEmission;
10 origMaxEmission=particleEmitter.maxEmission;
11 particleEmitter.emit=false;
12
13 }
14
15 function Update () {
16 if((delayTime+delayPlusTime)>0) delayTime-=Time.deltaTime;
17
18
19 if(delayTime<=0 && particleEmitter.emit==false) particleEmitter.emit=true;
20
21
22 if((delayTime+delayPlusTime)<=0){
23 particleEmitter.minEmission=origMinEmission*rampDownTime;
24 particleEmitter.maxEmission=origMaxEmission*rampDownTime;
25 rampDownTime-=Time.deltaTime;
26 if(rampDownTime<0){ rampDownTime=0;}
27 }
28
29 }
```

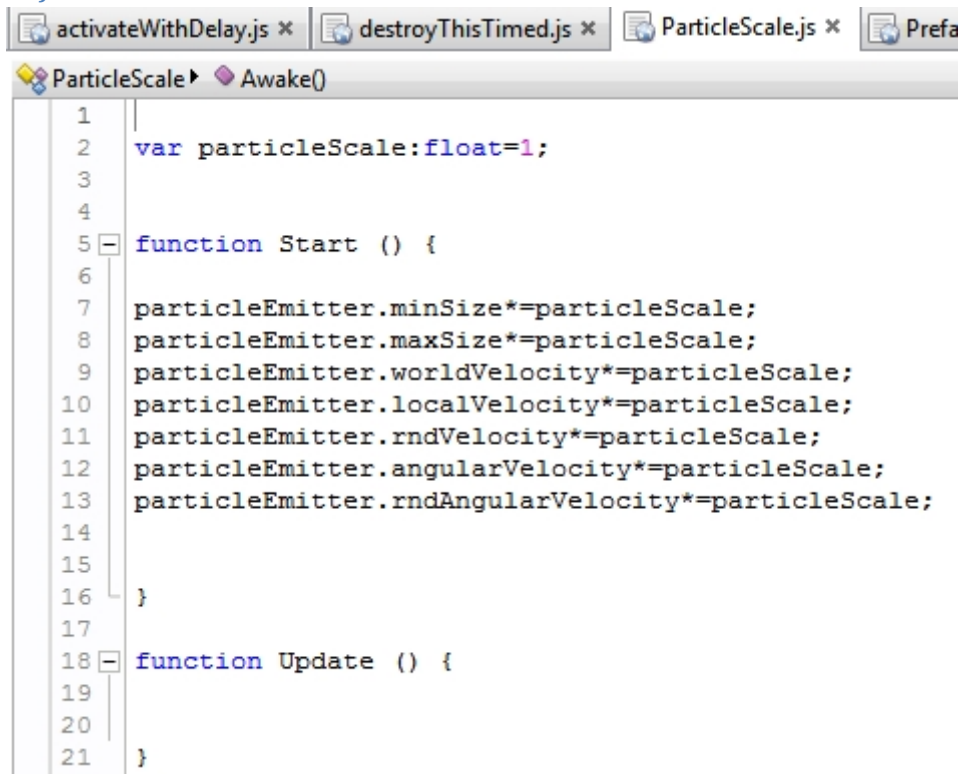
activateWithDelay.js

```
activateWithDelay ▶ Update()
1  var activateThis:GameObject;
2  var delay:float=2;
3  private var delayTime:float=0;
4
5  function Update () {
6  delayTime+=Time.deltaTime;
7  if(delayTime>delay) activateThis.SetActiveRecursively(true);
8
9  }
```

destroyThisTimed.js

```
destroyThisTimed ▶ Awake()
1
2  var destroyTime:float=5;
3
4  function Start () {
5  Destroy (gameObject, destroyTime);
6  }
7  function Update () {
8  }
```

ParticleScale.js



```
1 |
2 | var particleScale:float=1;
3 |
4 |
5 | - function Start () {
6 |
7 |     particleEmitter.minSize*=particleScale;
8 |     particleEmitter.maxSize*=particleScale;
9 |     particleEmitter.worldVelocity*=particleScale;
10 |    particleEmitter.localVelocity*=particleScale;
11 |    particleEmitter.rndVelocity*=particleScale;
12 |    particleEmitter.angularVelocity*=particleScale;
13 |    particleEmitter.rndAngularVelocity*=particleScale;
14 |
15 |
16 | }
17 |
18 | - function Update () {
19 |
20 |
21 | }
```

RayCast.js

```
activateWithDelay.js × destroyThisTimed.js × ParticleScale.js × PrefabGenerator.js × rampDownParticle.js × Raycast.js* ×
Raycast ▶ Update()
1 var moveThis : GameObject;
2 var hit : RaycastHit;
3 var createThis : GameObject[];
4 var cooldown : float;
5 var changeCooldown : float;
6 var selected:int=0;
7 var writeThis:GUIText;
8 private var rndNr:float;
9 function Start () {
10 writeThis.text=selected.ToString();
11 }
12
13 function Update ()
14 {
15     if(cooldown>0){cooldown-=Time.deltaTime;}
16     if(changeCooldown>0){changeCooldown-=Time.deltaTime;}
17     var ray = Camera.main.ScreenPointToRay (Input.mousePosition);
18     if (Physics.Raycast (ray, hit))
19     {
20         // Create a particle if hit
21         moveThis.transform.position=hit.point;
22
23         if(Input.GetMouseButton(0)&&cooldown<=0)
24         {
25             Instantiate(createThis[selected], moveThis.transform.position, moveThis.transform.rotation);
26
27             cooldown=0.1;
28         }
29     }
30     if (Input.GetKeyDown("space") && changeCooldown<=0)
31     {
32         selected+=1;
33         if(selected>(createThis.length-1)) {selected=0;}
34
35         writeThis.text=selected.ToString();
36         changeCooldown=0.1;
37     }
38
39     if (Input.GetKeyDown(KeyCode.UpArrow) && changeCooldown<=0)
40     {
41         selected+=1;
42         if(selected>(createThis.length-1)) {selected=0;}
43
44         writeThis.text=selected.ToString();
45         changeCooldown=0.1;
46     }
47
48     if (Input.GetKeyDown(KeyCode.DownArrow) && changeCooldown<=0)
49     {
50         selected-=1;
51         if(selected<0) {selected=createThis.length-1;}
52
53         writeThis.text=selected.ToString();
54         changeCooldown=0.1;
55     }
56 }
```

PrefabGenerator.js

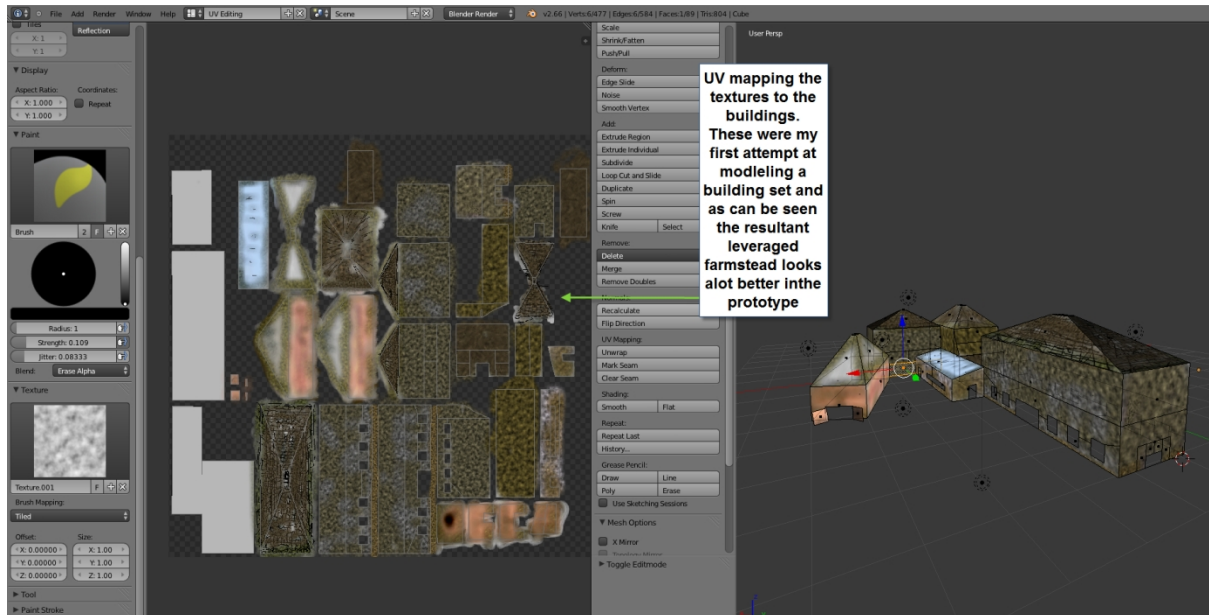
```
activateWithDelay.js x destroyThisTimed.js x ParticleScale.js x PrefabGenerator.js* x rampDownParticle.js x Raycast.js* x
PrefabGenerator ▶ Update()
1 var createThis:GameObject[]; // list of possible prefabs
2
3 private var rndNr:float; // this is for just a random number holder when we need it
4 var thisManyTimes:int=3;
5 var overThisTime:float=1.0;
6 var xWidth:float; /// define the square where prefabs will be generated
7 var yWidth:float;
8 var zWidth:float;
9 var xRotMax:float; /// define maximum rotation of each prefab
10 var yRotMax:float=180;
11 var zRotMax:float;
12 var allUseSameRotation:boolean=false;
13 private var allRotationDecided:boolean=false;
14 var detachToWorld:boolean=true;
15 private var x_cur:float; /// these are used in the random palcement process
16 private var y_cur:float;
17 private var z_cur:float;
18 private var xRotCur:float; /// these are used in the random protation process
19 private var yRotCur:float;
20 private var zRotCur:float;
21 private var timeCounter:float; /// counts the time :p
22 private var effectCounter:int; /// you will guess ti
23 private var trigger:float; /// trigger: at which intervals should we generate a particle
24
25
26
27 function Start () {
28 trigger=overThisTime/thisManyTimes; ///defines the intervals of time of the prefab generation.
29 }
30 function Update () {
31 timeCounter+=Time.deltaTime;
32 if(timeCounter>trigger&&effectCounter<=thisManyTimes)
33 {
34 rndNr=Mathf.Floor(Random.value*createThis.length); ///decide which prefab to create
35 x_cur=transform.position.x+(Random.value*xWidth)-(xWidth*0.5); /// decide an actual place
36 y_cur=transform.position.y+(Random.value*yWidth)-(yWidth*0.5);
37 z_cur=transform.position.z+(Random.value*zWidth)-(zWidth*0.5);
38
39 if(allUseSameRotation==false||allRotationDecided==false) /// basically this plays only
40 ///once if allRotationDecided=true, otherwise it plays all the time
41 {
42 xRotCur=transform.rotation.x+(Random.value*xRotMax*2)-(xRotMax); /// decide rotation
43 yRotCur=transform.rotation.y+(Random.value*yRotMax*2)-(yRotMax);
44 zRotCur=transform.rotation.z+(Random.value*zRotMax*2)-(zRotMax);
45 allRotationDecided=true;
46 }
47 ///var justCreated:GameObject=Instantiate(createThis[rndNr], Vector3(x_cur, y_cur, z_cur), Vector3(xRotCur, yRotCur, zRotCur));
48 ///instantiates the prefab
49 var justCreated:GameObject=Instantiate(createThis[rndNr], Vector3(x_cur, y_cur, z_cur), transform.rotation);
50 ///create the prefab
51 justCreated.transform.Rotate(xRotCur, yRotCur, zRotCur);
52
53 if(detachToWorld==false) /// we attach the freshly generated prefab to the object that is holding this script (if needed)
54 {
55 justCreated.transform.parent=transform;
56 }
57 timeCounter-=trigger; ///administration :p
58 effectCounter+=1;
59 }
60 }
```

END OF GAME CODE

Appendix E

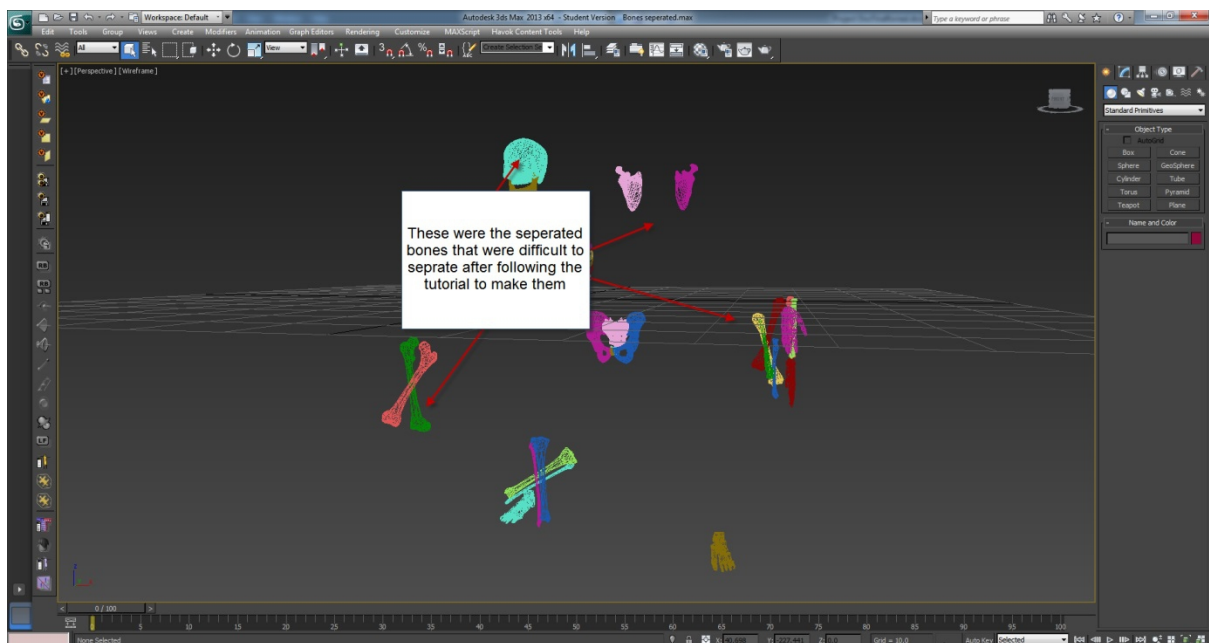
Other software undertaken to complete the project

Blender 2.6.4

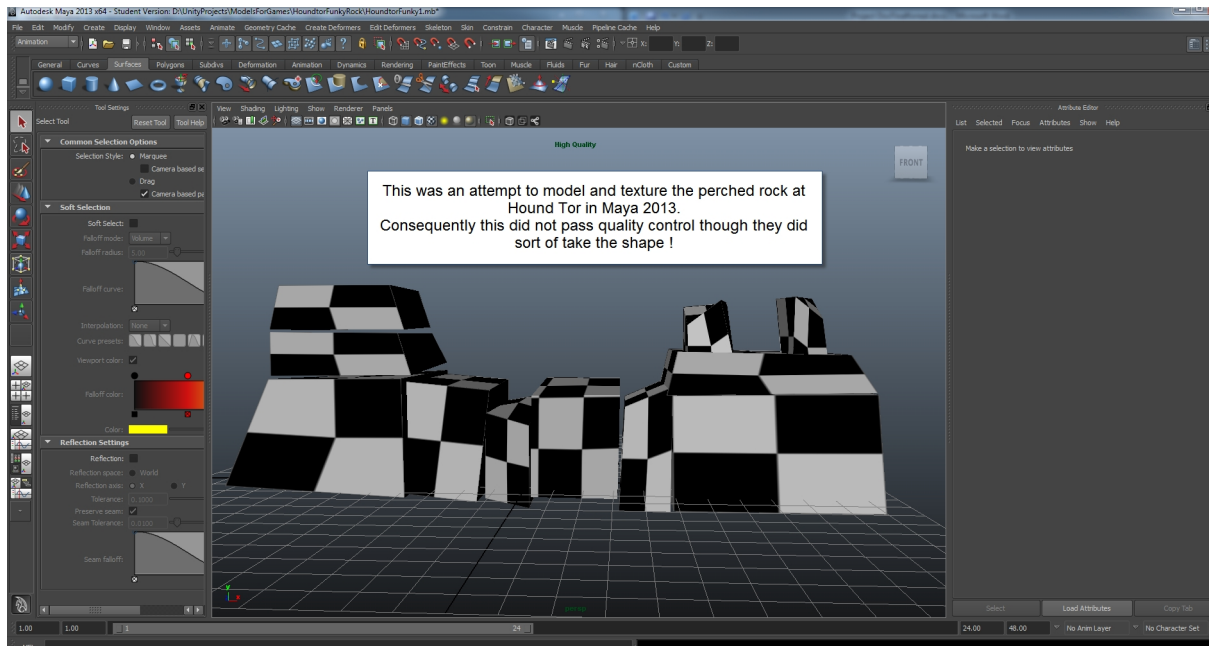


Autodesk Software 2013

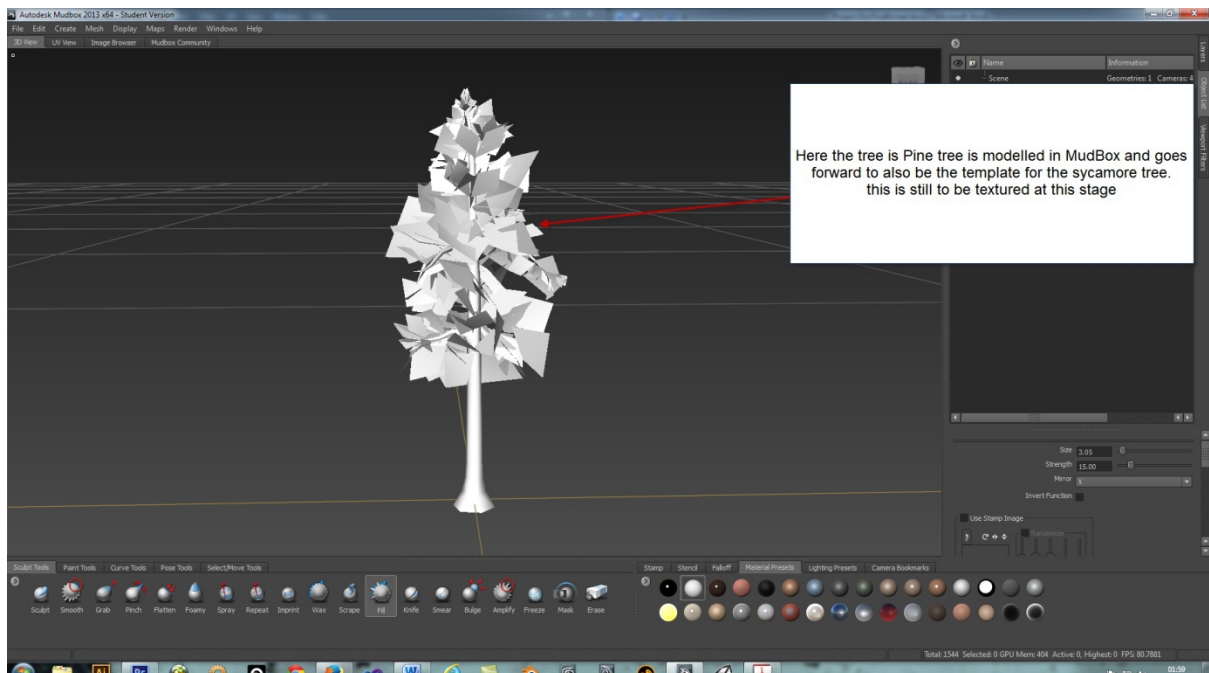
3DS Max



Maya 2013



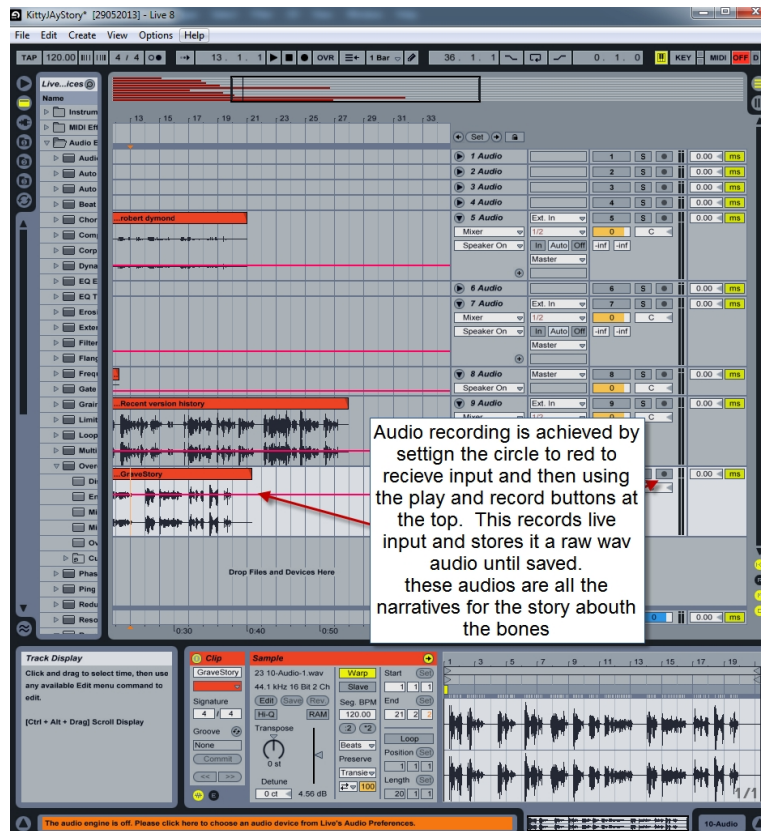
MudBox 2013



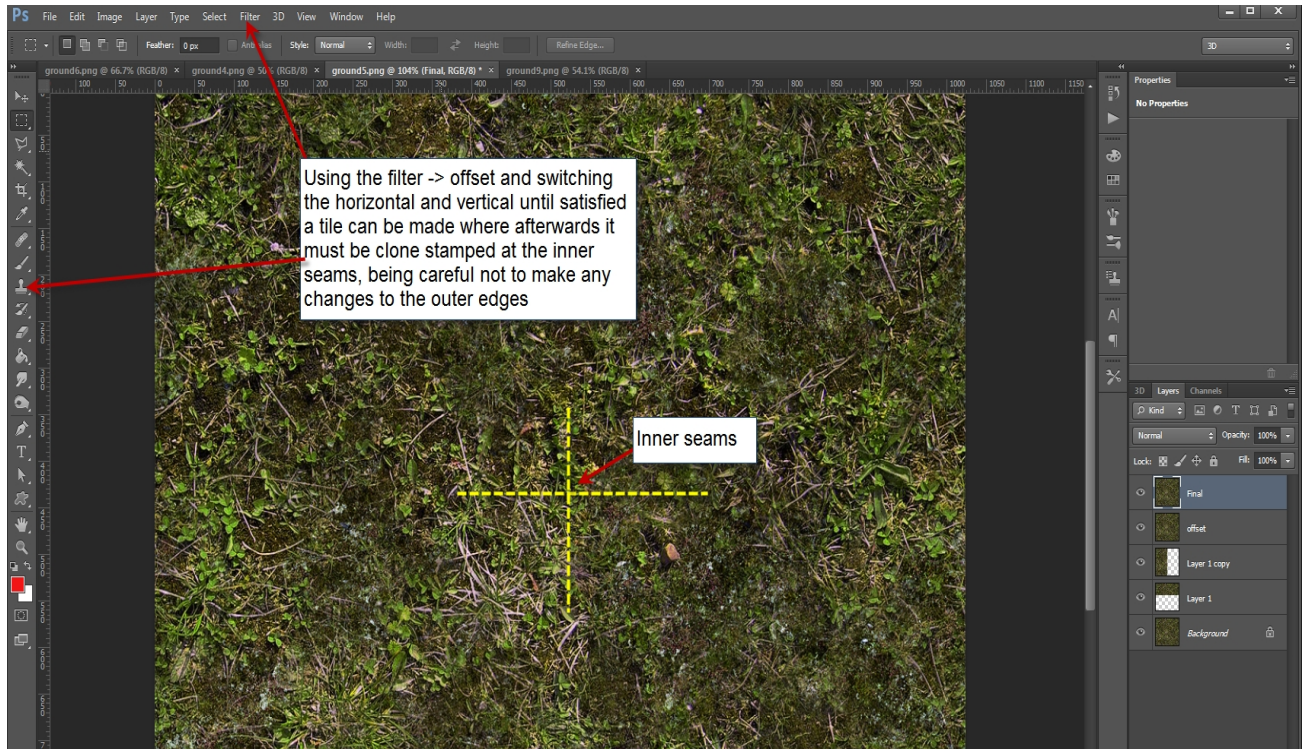
FL Studio 10



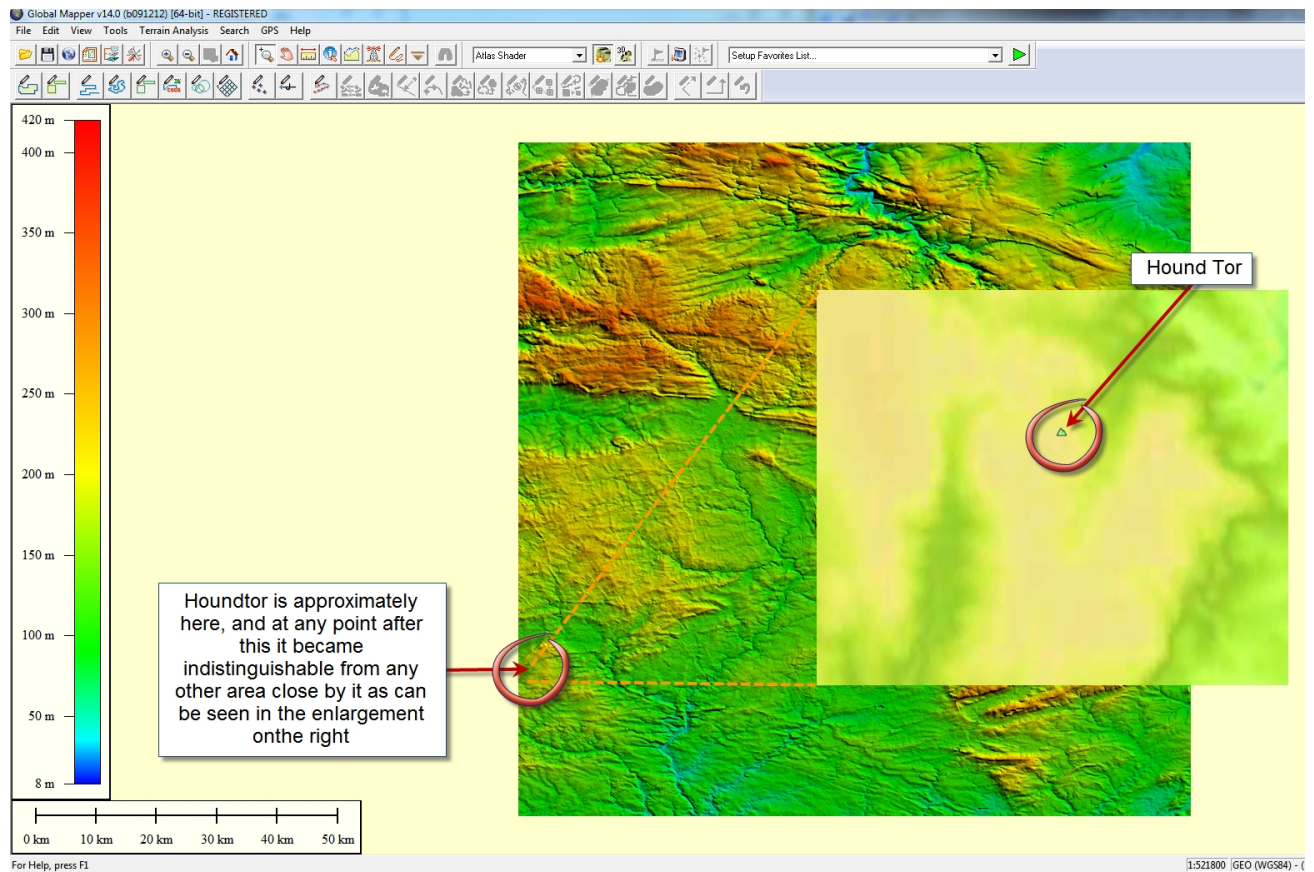
Abelton Live 8



Adobe PhotoShop CS5



Global Mapper 14



Appendix F

These can be found in Appendices under **Appendix A** in the second part Project report Deliverable.

Appendix F

